THE UNIVERSITY
of EDINBURGH

Multi-Period Sales Districting Problem

Saranthorn Phusingha

Doctor of Philosophy
University of Edinburgh
2020

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Saranthorn Phusingha, May 2021*)

To my lovely family

Acknowledgements

First, I would like to express my gratitude to the Development and Promotion of Science and Technology Talents Project (DPST) for financial support during my MSc and PhD study. Without this scholarship, I would not have had the opportunity to study abroad and broaden my horizons.

Next, my deepest appreciation goes to Dr.Joerg Kalcsics for being such a great supervisor. Your generous guidance and effort have contributed enormously to this work. Thanks very much for teaching me how to write good work. More importantly, thanks for being a lovely and supportive brother.

My work was greatly improved with the help of several people. Paula Fermín Cueto, thank you for your valuable comments which helped to improve Chapters 1 and 2. Thank you so much for drawing the examples in Chapter 1. Tom and Helen Byrne, you are fantastic proofreaders. This work would contain many more horrible grammatical errors without your help. Moreover, I wholeheartedly appreciated feedback from Akshay Gupte and Daniel Gartner, my internal and external examiners. I was grateful for your valuable suggestions during the corrections.

My life in Edinburgh would be really boring without these friends: Minerva, Tom, Maria, Marion, Ivet, Rodrigo, Wenyi, Xavier, Ivona, Paula, Nagisa, Shunee, and Julian. Thank you all for every fun moment during my PhD. Your friendship means a lot to me. Moreover, I would like to thank the Edinburgh Research Group in Optimization (ERGO) for valuable academic experience during my studies. Also, I would like to thank my family and friends in Thailand. I really appreciate your support and encouragement, especially from my Mum, my sister, and my brother who always trust in me and support my decisions. Moreover, I would like to thank my Dad who became an angel in heaven. I am sure that you are always proud of me.

I am deeply grateful to Ian and Elaine Auld. Thank you for providing me with a new sweet home and accepting me as a new member of your lovely family. Finally, I would like to show my greatest appreciation to Graeme Auld, my amazing boyfriend. I really appreciate your time helping me proofread this thesis before submission. Also, thank you for comforting me whenever I feel depressed and encouraging me to continue doing my best. Every day is a special day, even during the most exhausting time from writing a thesis, because of you. Thank you for being a part of my life.

Lay Summary

Sales promotion is one of the most crucial strategies to boost a business. Most manufacturing and consumer product companies usually send salesmen to visit their customers' sites regularly to promote sales and provide information about their new product range. Typically, a salesman is assigned to a specific group of customers for the long term, as they can build personal relationships with customers, which is beneficial to customers' satisfaction. Since a salesman has to travel to their customers' sites, the location of customers mainly affects their working performance. Suppose the locations of customers to get a service on the same day are significantly far from each other. In that case, a salesman tends to waste a significant amount of time travelling from one place to another, resulting in low productivity and morale. Eventually, this affects overall sales performance and company profit. Therefore, the decisions on allocating customers for each salesman are of paramount importance.

One way to reduce unnecessary travel time is to allocate customers who are geographically close to each other to the same salesman. At the same time, the total workload arising from providing the service should be approximately equal for each salesman to promote fairness. In this case, each customer requires recurring services with different visiting frequencies such as every week or two weeks during a fixed planning horizon. Therefore, valid visiting schedules for the salesmen corresponding to the customers' visiting requirements are also required. This kind of requirement can be seen in regular engineering maintenance or sales promotion by manufacturers. Here, we aim for weekly schedules such that each week has an almost equal weekly workload and customers who get service in the same week are geographically close to each other. The latter provides more flexibility to modify the plan of visiting for salesmen, without increasing the travel time too much compared to the original plan. This is advantageous to salesmen especially when they have essential reasons to postpone a visit to another day.

Although the geographical proximity of customers has been considered in several studies for the problem of allocating customers to each salesman, it has not been considered as useful information for creating schedules. Therefore, we first focus on creating effective schedules for a salesman where the set of their customers is known. We propose two solution approaches: Benders' decomposition and tabu search. Benders' decomposition is a solution approach that can find the optimal solution, while tabu search is a method that can find a good solution quickly but is not guaranteed to find the optimal one. We develop the solution approaches by employing sophisticated techniques and test them separately on small data instances that contain $30-50$ customers. The results show that Benders' decomposition struggles when the number of customers increases. On the other hand, tabu search performs well in every case.

Due to the success of tabu search in finding effective schedules, we extend the method to solve the whole problem, i.e., partitioning customers to each salesman and determining effective schedules at the same time. The extended method is successful in solving instances of $100-300$ customers.

This research shows the challenge of partitioning customers and deriving high-quality schedules simultaneously, so further studies, including the creation of more effective solution approaches, are still required.

Abstract

In the sales districting problem, we are given a set of customers and a set of salesmen in some area. The salesmen have to provide services at the customers' locations to satisfy their requirements. The task is to allocate each customer to one salesman, which partitions the set of customers into subsets, called districts. Each district is expected to have approximately equal workload and travel time for each salesman to promote fairness among them. Also, the districts should be geographically compact since they are more likely to reduce unnecessary travel time, which is desirable for economic reasons. Moreover, each customer can require recurring services with different visiting frequencies such as every week or two weeks during a planning horizon. This problem is called the 'Multi-Period Sales Districting Problem (MPSDP)' and can be found typically in regular engineering maintenance and sales promotion.

In addition to determining the sales districts, we also want to get valid weekly visiting schedules for the salesmen corresponding to the customers' visiting requirements. The schedules should result in weekly districts with the following desirable characteristics: each weekly district should be balanced in weekly workload and geographically compact. The compactness in the schedules provides benefits when a salesman has to deal with short-term requests from customers or change a visiting plan during the week. Namely, the salesman can postpone a visit to another day if necessary, without increasing the travel time too much compared to the original schedule. This is beneficial when the salesman has to deal with unexpected situations, for example, road maintenance, traffic jams, or short notice of time windows from customers. Although the problem is very practical, it has been studied only recently. Since most of the previous literature on general scheduling problems did not consider compactness, a few recent studies have begun to focus on solving the scheduling part of the problem.

The purpose of this research is to develop a more sophisticated exact solution approach as well as an efficient high-quality heuristic to solve the scheduling part. Eventually, with an effective elaborate method to solve the scheduling part, we aim for a robust algorithm to solve the districting and scheduling part of the problem simultaneously.

This thesis contains three main parts. The first part introduces the problem and provides a mixed-integer linear programming formulation for only the scheduling part and formulation for the whole problem. The second part presents solution approaches, including an exact method and a heuristic, for only the scheduling part. The last part is dedicated to further development of a successful approach from the second part to solve the districting and scheduling part of the problem simultaneously.

For solving the scheduling part, Benders' decomposition is developed as a new exact solution method. The linear relaxation of the problem is strengthened by adding several Benders' cuts derived from fractional solutions at the beginning of the algorithm. Moreover, a good-quality integer solution derived from a location-allocation heuristic is used to generate cuts beforehand, which significantly improves the upper bound of the objective function value. Nondominated optimality cuts are implemented to guarantee the strongest Benders' cuts in each iteration. Also, instead of generating a Benders' cut per iteration, we exploit the decomposable structure of the problem formulation to generate multiple cuts per iteration, resulting in a noticeable improvement in the lower bound of the objective function value. In the classical Benders' decomposition, one of the main factors that slow down the algorithm is that one has to solve the integer programmes from scratch in each iteration. To alleviate this problem, a modern implementation creates only one branch-and-bound tree and adds Benders' cuts derived from a solution in each node in a solution cut pool. This method is called branch-and-Benders' cut. To assess the suitability of the algorithm, we compare its performance on small data instances that

contain 30−50 customers to the Benders' algorithm in CPLEX and show that our algorithm is highly competitive.

Since an exact solution method usually struggles to solve realistic large data instances, a meta-heuristic called tabu search is proposed. A high-quality initial solution to start the algorithm is derived from the location-allocation heuristic. Three different neighbourhoods based on information about week centres or customers' week patterns are created within which we search for the best solution. An infeasible solution is allowed in the search to expand the search space. During the search, the size of a whole neighbourhood can be excessively large, so we limit the search to promising areas of the solution space to save computational time. Also, a surrogate objective value is used to save on computational time in cases when computing the real objective value is too time-consuming. Although the tabu search defines a list of forbidden moves to avoid the cycle of solutions, the algorithm can still struggle to avoid being trapped around a local optimum. Therefore, a diversification scheme is proposed for such cases. The algorithm is also accelerated by combining all neighbourhoods and selecting the appropriate neighbourhood for each iteration by a roulette wheel selection. It shows impressive results in small data instances that contain 30−50 customers. The comparison with built-in heuristics in CPLEX confirms the robustness of the tabu search algorithm. Finally, we combine the tabu search algorithm with our developed Benders' decomposition. Numerical results show that the tabu search method improves the upper bound of the Benders' decomposition algorithm. However, the overall performance is not satisfying so the combination of these two techniques still requires more proper development.

As the tabu search algorithm performs well on the scheduling part, it is extended to solve the whole problem, i.e., the districting and scheduling part at the same time. Computational results on large data instances, which contain between 100 and 300 customers, demonstrate its capacity to derive a high-quality solution within a reasonable amount of time, i.e., less than 17 minutes. At the same time, the Benders' decomposition algorithm in CPLEX, which is a benchmark in this case, and the built-in heuristics in CPLEX cannot even find any feasible integer solution for most of the instances within an hour. Importantly, there is a conflict between the districting part and the scheduling part so we recommend solving both parts simultaneously for tackling the MPSDP.

The multi-period sales districting problem is highly practical and challenging to solve. To the best of our knowledge, we are the first to propose a single integrated solution approach to solve the whole problem. Further studies including adding more realistic planning requirements into consideration and effective solution approaches to solve the problem are still required.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

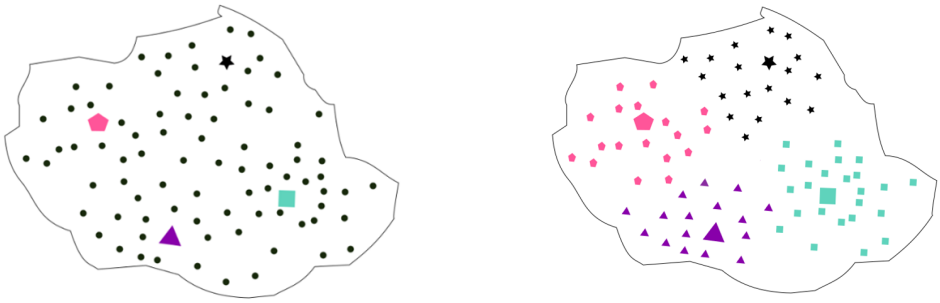
Introduction

In this chapter, Section 1.1 introduces districting problems and some of the challenges on how to formulate them. Also, in Section 1.1, we present one of the applications of districting problems, the *classical sales districting problem*, which is the precursor of the multi-period sales districting problem. Prominent solution approaches to solve districting problems are also included. Finally, the outline of the thesis is described in Section 1.2.

1.1 Districting Problems

According to Kalcsics & Ríos-Mercado (2019), districting is a problem that attempts to partition a given set of small units, called *basic units*, into several larger groups of basic units called *districts*. Examples of basic units are customers, streets, or zip code areas.

Figure 1.1 shows an example where the basic units are customers. In Figure 1.1a, the enlarged symbols and the black dots represent, respectively, offices of salesmen and customers. We aim to allocate customers to each salesman. As a result, a set of customers that are served by the same salesman is a district in this context. Figure 1.1b shows four districts where the small symbols represent customers in a corresponding district. Other examples of districts where basic units are streets and zip code areas are presented in Figure 1.2, which are from Figure 25.1 in Kalcsics & Ríos-Mercado (2019).



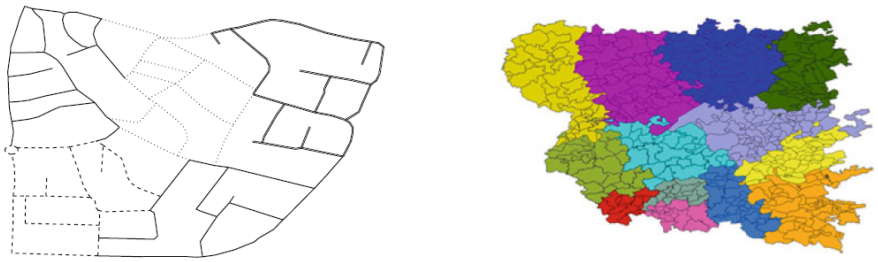
(a) Before partitioning basic units into districts

(b) After partitioning basic units into districts

Figure 1.1: An example of districts where basic units are customers.

Districting problems have been studied extensively and Figure 1.3 shows the number of papers related to districting problems each year from Scopus.

Districting problems are more advanced than general clustering problems that only aim to group similar objects since all districts are supposed to satisfy the following three attributes: *balance*, *compactness*, and *contiguity*. First, balance is an attribute that says each district should have an approximately equal size, where the specific size measurement depends on the context of each problem. Second, compactness indicates that the geographical distance among



(a) Districts for streets (b) Districts for zip code areas

Figure 1.2: More examples of districts [Figure 25.1 in Kalcsics & Ríos-Mercado (2019)].

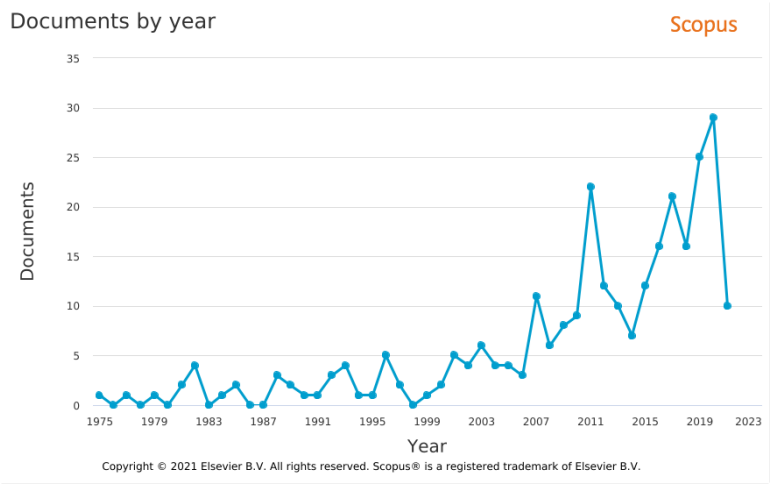


Figure 1.3: Annual trend of districting problems according to https://www.scopus.com.

basic units in each district is not too large. Note that a compact district tends to be round-shaped. Finally, contiguity describes whether it is possible to travel between all basic units of the district without having to pass through other districts, i.e., distinct districts should not overlap or at least have a small overlap. Each attribute provides different benefits depending on the specific applications under consideration. Figure 1.1b is an example of highly contiguous and compact districts. We also provide examples of districts that are low in either contiguity or compactness in Figure 1.4. Figure 1.4a shows districts that are compact but low in contiguity, as their basic units overlap in the middle of the area. In contrast, Figure 1.4b presents districts that are contiguous but low in compactness: see the distorted shape of every district.

(a) Low contiguity (b) Low compactness

Figure 1.4: Examples of districts that are low in either contiguity or compactness [Drawn by Paula Fermín Cueto].

Unfortunately, ways of quantifying the desired attributes of districts are unclear and usually specific to the context of problems, especially the geometric representation of the basic units.

As a result, districting problems do not have a unique model formulation. Here we will provide a brief overview of the limitation and difficulties relating to the measurement of each attribute. For more details, we refer the interested reader to Kalcsics & Ríos-Mercado (2019).

Basic units are usually allocated to only one district so it is challenging to guarantee that every district will have the same size. Therefore, common approaches to measuring the local balance are related to the imbalance of districts. The first approach is to evaluate the deviation from the mean of the district size (Ríos-Mercado & Fernández 2009, de Assis et al. 2014). The second approach is to define an acceptable range of the district size, for example, Bozkaya et al. (2011). These bounds can be derived from a defined acceptable deviation level from the mean of the district size, or any restrictions related to the problem, for example, the limited working time. In terms of the global balance, it is usually derived from the maximum balance of a district. Another option is from the sum of balance from every district (Bozkaya et al. 2003). These two options for the global balance have a disadvantage; the maximum balance does not include the information of every district, while the sum balance is at risk of bias since well-balanced districts might compensate for noticeably unbalanced districts. Therefore, another possible approach is to consider the convex combination of these two options (Butsch et al. 2014).

The ways to define balance are easily derived as linear expressions. However, it is not clear how to treat the balance attribute in a model formulation. Some works treat the balance as a hard constraint in form of an acceptable range (Fleischmann & Paraschis 1988, Zoltners & Sinha 2005); others set balance as the objective function (Blais et al. 2003, de Assis et al. 2014).

Regarding contiguity and compactness, they do not have rigid mathematical definitions. In particular, they mostly depend on the geometric representation of basic units, e.g., points, polygons, or lines.

In terms of contiguity, it is easy to define neighbourhoods where the basic units are lines or polygons. For example, if the basic units are polygons, they belong to the same neighbourhood if they share a common border. In particular, it is easy to identify the connectedness among basic units. Contiguity in such cases is usually defined as a hard constraint (Drexl & Haase 1999, Shirabe 2009).

However, it is ambiguous and difficult to measure contiguity when the basic units are represented by points since it is not clear how to define the shape of districts. A possible approach is to define a proximity graph to approximately represent the adjacency of the basic units, e.g., Gabriel graph or Voronoi diagram (Gross & Yellen 2003, Lei et al. 2012). Nevertheless, this approach is not popular and, instead, most previous studies ignore contiguity and consider a suitable compactness measure to get a small amount of overlap between districts.

Regarding compactness, measures are specific to the geometric types of basic units. For basic units that are polygons, the local compactness usually relies on the area or the perimeter length of districts. This geometric measure, however, is not applicable for basic units that are lines or points.

For basic units that are lines or points, a distance-based measurement is popular. This is also suitable when travelling within districts is a key component of the problem, e.g., mail service or salesman visiting customers. The most common approach is a centre-based measure which is derived from the sum of the distance between every basic unit in the district and the district centre. Note that the distances can be squared or weighted. There are other ways to measure the local compactness that are more intuitive such as pairwise distances, i.e., the sum of distances between each pair of basic units. This can lead to difficulties in solving the model (Bender et al. 2016). Instead of summation, it is possible to consider the local compactness from the maximum distance from a district centre or among basic units (Ríos-Mercado & Fernández 2009, Elizondo-Amaya et al. 2014, Rios-Mercado & Escalante 2016). However, this is much less common. In terms of global compactness, it is usually either a summation or the maximum of the local compactness of every district.

As we discussed before, when basic units are points, the majority of works in such cases rely on the compactness measurement to get a small overlap among districts. A way to select such a proper compactness measure, however, is challenging since each compactness measure can lead to a different level of contiguity in different problems. Also, several factors influence how much overlap among districts there is. The distribution of the district centres is also a key factor. If the centres are close to each other, the districts are at risk of overlapping. Also, a

strict acceptable range for the balance tends to result in more overlapping districts.

Most model formulations of districting problems set compactness as the objective function. The distanced-based measurement provides a benefit in this case as it is easy to express in a linear form, or a quadratic form in the case of the pairwise-distanced approach.

The ways to measure the desired quantities are strongly dependent on the context of the problem so there is no unique mathematical model formulation for districting problems. As the particular application of districting problems plays a crucial role in the design of model formulations, we will present the main applications of districting problems in the next section. One of the applications is the classical sales districting problem which is the precursor of our problem.

1.1.1 Application: Classical Sales Districting Problem

Districting problems can be categorised into four major types corresponding to their areas of applications: *sales districting*, *service districting*, *distribution districting* and *political districting*. Sales districting problems focus on assigning customers who require service at their locations to each salesman. Service districting problems can be classified into two main types. The first type relates to allocating residential areas to shared social facilities, for example, hospitals and schools. The second type partitions areas in such a way that each household in the same area receive services such as postal delivery and bin collection service from the same workers. Distribution districting problems divide up areas for pickup and delivery services. Finally, political districting problems aim to determine electoral constituencies in a city.

This research focuses on an extension of the classical sales districting problem. More details of the other three applications in terms of problem formulations and proposed solution approaches can be found in Kalcsics & Ríos-Mercado (2019).

Problem Statement

The classical sales districting problem consists of a given set of customers and salesmen. Customers are located in some specific area and require services from salesmen at their locations. Salesmen travel from their office to provide services within the limitation of their working time. The task in this problem is to allocate customers to each salesman so that each customer is served by a unique salesman. As a result, each salesman is assigned a set of customers or a district to work in. Each district is supposed to have the desired attributes that were mentioned earlier since they are beneficial to the overall performance of the salesmen. Balance in workload, which is calculated from total service time and travel time, enhance fairness among salesmen. Contiguity guarantees non-overlapping districts which can avoid competition between salesmen for customers with a high sales potential (Kalcsics & Ríos-Mercado 2019). Moreover, contiguous and compact districts are believed to reduce unnecessary travel time, leading to more productive work for salesmen (Zoltners & Sinha 2005, Kalcsics & Ríos-Mercado 2019). Zoltners & Sinha (2005) convincingly claimed that compact, contiguous and balanced districts could increase profit for a company due to better utilisation of work time by salesmen.

However, there are difficulties with measuring these desirable attributes. Regarding the balancing criterion, service time and travel time should be balanced for every salesman. However, exact travel time is intractable and costly to compute in reality and it is affected by many complicating factors. For example, some customers may require an exact time to be visited; or some unexpected occasions such as road maintenance or traffic jams can delay a service. Therefore, the travel time is usually approximated in practice (Bard & Jarrah 2009, Lei et al. 2012, 2015). Another common approach is to rely on compactness and contiguity instead since these attributes directly influence travel time. One of the most prominent ways to reduce unproductive travel time is to maximise compactness and ensure contiguity. Unfortunately, both of these attributes do not have rigorous expressions since basic units are represented by points. As we discussed before, most previous studies focus on a suitable compactness measure to get fewer overlapping districts. In this context, the most popular measurement for compactness is a centre-based measure where the district centres in this case are usually the offices of salesmen.

Mathematical Formulation

Due to ambiguity in how to quantify these attributes, there are varieties of proposed mathematical formulations for districting problems. The most common model for sales districting problems is to maximise the compactness of all districts while ensuring that districts are balanced. Note that a typical way to maximise compactness is to minimise the total distance-based measurement. This model was first proposed by Hess et al. (1965) to solve a political districting problem. For a given set of voters, the problem aims to derive balanced districts in the number of voters. Usually, the number of districts is predefined. For calculating compactness, the centre of each district is selected among the voters, and then the centre-based distance is computed. In particular, the decisions in the problem concern not only partitioning the voters but also selecting the proper district centres for measuring the compactness. Hess & Samuels (1971) adapted the model of Hess et al. (1965) for the sales districting problem by replacing voters with customers and keeping balance in terms of total service time in each district instead.

There are a few different proposed goals in mathematical models for the classical problems. Since the primary goal for each company is to get the largest profit, some studies aimed to maximise profit by assigning customers to each salesman and also the optimal service time (for example, Lodish (1975)). However, Zoltners & Sinha (2005) stated that these two problems should be separately considered since they are revised in a different period. The districting problem is considered for the long term. In contrast, the time allocation is required to be updated more frequently, as it can be influenced by short-time effects, for example, sales of beverages in different weather. Some previous studies defined the objective function by maximising profit and ignoring balance in districts. Drexl & Haase (1999) implemented a mathematical model for a beverage company in Germany and showed good computational results. However, Zoltners & Sinha (2005) argued that it is hard in practice since profit depends on many aspects which are tricky to include entirely in one model, and salesmen usually prefer workload balance.

1.1.2 Solution Approaches

Many solution approaches, including exact methods and heuristics, have been proposed for districting problems. Some examples are summarised in Table 1.1.

As an exact solution approach usually struggles to solve large data instances, heuristics are attractive to solve these cases. One of the most well-known heuristics is a location-allocation heuristic adapted from Cooper (1964). The method was applied to solve the mathematical formulation in Hess et al. (1965). It divides the problem into two phases. The first phase, the location phase, consists of finding the location of suitable district centres for given districts. Then, with the fixed district centres from the location phase, customers are allocated to the district centres while ensuring the balancing criterion in the allocation phase. These two phases are solved sequentially in an alternating manner until both of them reach the same objective value. This technique is effective and has been successfully applied in several studies.

According to Kalcsics & Ríos-Mercado (2019), meta-heuristics have gained more attention recently due to their flexibility in dealing with complicated constraints, for example, the Greedy Randomised Adaptive Search Procedure (GRASP), tabu search, and genetic algorithm. More details can be found in Kalcsics & Ríos-Mercado (2019).

1.2 Outline of Thesis

In Chapter 2, we introduce an extension of the sales districting problem called the multi-period sales districting problem, including the literature review and model formulations. Then, we propose sophisticated solution approaches for solving only the scheduling part of the problem. The methods include Benders' decomposition as an exact solution approach and tabu search as a meta-heuristic, presented in Chapter 3 and 4, respectively. As the tabu search algorithm proves highly successful in solving the scheduling part, we extend the method to solve both parts of the problem at the same time: see Chapter 5. Finally, Chapter 6 provides the conclusion and further study.

Catagory	Method	Reference
Exact solution approach	Branch-and-price algorithm	Mehrotra et al. (1998) Bender et al. (2018)
	Branch-and-bound combined with a cut generation strategy	Salazar-Aguilar et al. (2011)
Heuristic	Location-allocation heuristic	Hess et al. (1965) Bender et al. (2016)
Meta-heuristic	GRASP	Ríos-Mercado & Salazar-Acosta (2011) Salazar-Aguilar et al. (2013)
	Tabu search	Bozkaya et al. (2003) Haugland et al. (2007)
	Simulated Annealing	D'Amico et al. (2002)
	Genetic algorithm	Tavares-Pereira et al. (2007) Steiner et al. (2015)

Table 1.1: Examples of solution methods in districting problems.

Chapter 2

Multi-Period Sales Districting Problem (MPSDP)

The problem considered in this thesis is an extension of the classical sales districting problem called the *Multi-Period Sales Districting Problem* (MPSDP), which was first introduced by Bender et al. (2016). This problem is much more complicated and realistic than the classical problem. The classical problem is considered in only one period, while this new problem covers multiple periods. Moreover, the MPSDP includes a regular requirement from customers which, however, has only been studied recently, namely, obtaining recurring service with a specific preference of frequency. In this case, customers may require service every week, every two weeks, or even every month. This kind of problem can be seen in regular sales promotion at customers' sites or engineering field maintenance (Bender et al. 2016).

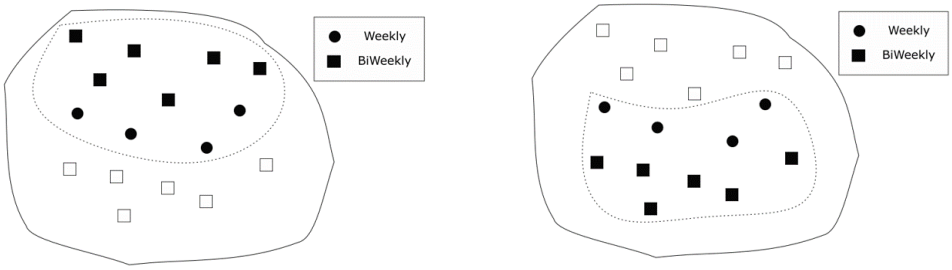
2.1 Problem Statement

The MPSDP aims to partition customers into districts and, at the same time, determine schedules for visiting customers that fulfil their frequency requirement. In this section, we will introduce the notation used to formulate the problem and present characteristics of schedules that provide benefit to the performance of salesmen.

In the problem, we consider a period in terms of weeks. The *planning horizon* is the total number of weeks that we are considering for the schedules. A required frequency to obtain service for each customer is called a *week rhythm*. Note that each customer can require a different frequency of visiting, e.g., every week or every month. A *visiting week* of a customer is a week that a salesman visits the customer. We call the combinations of visiting weeks corresponding to customers' week rhythms *week patterns*. For example, if we consider the planning horizon over eight weeks and a customer demands services every two weeks, then there are two week patterns; the first week pattern has week 1, 3, 5 and 7, while the other one has week 2, 4, 6, 8. Note that the number of week patterns corresponds to a week rhythm. The set of customers who belong to the same visiting week is called a *week cluster*.

Now, for the predefined planning horizon, the tasks of this work are not only to determine a responsible district for each salesman but also visiting weeks for customers that satisfies their week rhythms. In particular, we aim for schedules that result in balanced and compact week clusters. Balance helps to avoid overload of work during the planning horizon, while compactness is advantageous for the performance of salesmen since it can help to reduce unnecessary travel time each week. More importantly, a salesman can easily modify a plan to visit customers within a week, especially when unexpected situations happen, for example, road maintenance, traffic jams, or short notice of time windows from customers. In these cases, the salesman can postpone the visits to another day, without increasing the travel time significantly compared to the original plan (Bender et al. 2016). Examples of compact weekly schedules are presented in Figure 2.1, where Figures 2.1a and 2.1b show week clusters of customers who require service every week and every two weeks. However, Bender et al. (2016) warned that these desirable attributes also depend on how customers of different week rhythms spread across a district.

For example, if many customers who require weekly services are spread across a district, it is inevitable for a salesman to travel around the whole district every week.



(a) Week 1 (b) Week 2

Figure 2.1: Compact weekly schedules.

Apart from the weekly schedules, the problem also aims for daily schedules with the same desirable characteristic, i.e., compactness and balance. These attributes, especially compactness, are beneficial to salesmen during the day. The compactness in the daily schedules provides the flexibility in arranging a sequence of customers to get service on the same day. For example, customers might have a specific time window suddenly so that they have to meet a salesman later than the original schedule. The salesman can visit other customers first and then return to visit these customers later. In this case, he does not increase substantially time from the original plan since every customer who gets the service on the same day are supposed to be close to each other.

The daily schedules are more essential in some applications that customers may require more than one service per visiting week and have specific days to get the service. For example, replenishment of beverages or cigarettes in a machine usually requires more than one visit per week. Note that necessary notations for the daily schedules are analogous to the week level, namely, visiting days, weekday patterns and day clusters.

Designing daily routes is also a part of the planning. However, Bender et al. (2016) reported that some customers usually request changes of a visiting day to another day in the short term, resulting in a frequent update in the daily routing plan. Therefore, the authors suggested separating the routing plan from the main problem and solving it just whenever every operational detail is ready.

Although the MPSDP is very relevant in practice, it has only been studied recently, as we discuss in Section 2.2.

2.2 Literature Review

The classical sales districting problem has been the focus of substantial research in mathematical formulations and solution methods. However, the study of the MPSDP is still very scarce.

2.2.1 Mathematical Formulations

Prior to Bender et al. (2016), all previous studies of sales districting problems failed to address at least one crucial aspect of the MPSDP.

There are only a few studies of districting problems with multiple periods. Lei et al. (2015) presented a problem where customers require one visit from salesmen in each period of a planning horizon. In this case, a period consists of several weeks and every salesman travels from a depot that belongs to the set of available depots. The specific characteristic of the problem is that customers are changed dynamically and then revealed at the beginning of a period. In other words, the set of customers in every period is known in advance. The goal of this problem is to design districts in each period where each of the districts is then assigned to a salesman who stays in the closest depot. Every district is further partitioned into sub-districts for specific working days. Namely, a salesman visits customers in a sub-district once on a particular day

during the period. The objective function is to minimise a weighted sum of four measures: the number of districts, the sub-district compactness, the difference of districts in subsequent periods, and the variance on the average of salesman profit in each period. Note that the profit, in this case, is derived from the difference between revenue from visiting customers and the travel costs in the sub-districts. Although the authors considered compactness as one criterion for sub-districts, customers do not require recurring services with different frequencies over the planning horizon, which is an essential feature of the multi-period sales districting problem. Therefore, we cannot use the problem in Lei et al. (2015) to represent the MPSDP.

Multi-period scheduling problems can be found in various applications, for example, maintenance problems (Wei & Liu 1983), and logistics (Campbell & Hardin 2005, Kazan et al. 2012). A closely related problem to the MPSDP was presented by Núñez-del-Toro et al. (2016). The authors proposed a mathematical formulation for a multi-period service scheduling problem, where there are a given set of customers who require recurring service over a finite planning horizon and a set of operators which can serve a limited number of customers in each period. The task is then to allocate the customers to the operators, such that the customers' demands are satisfied and the capacity of each operator is not exceeded in each period. Usually, operators provide service in periods that strictly correspond to customers' week rhythms. In this work, the authors also considered the effect of relaxing customers' week rhythms, i.e., allowing an operator to serve its customers before the periods that customers request. They proposed mathematical formulations for serving customers in both cases: strict and relaxed customers' week rhythms. They formulated periodic services for customers by using binary variables to decide if customers will receive service in a week or not. Although this problem includes the requirement of the customers obtaining recurring service with different frequencies, the compactness of customers in each period is not taken into account. Moreover, the goal of this problem is to minimise the number of utilised operators along the planning horizon, which is entirely different from ours. Therefore, we cannot adapt their mathematical formulations to our problem.

Periodic vehicle routing problems have one common feature to our problem: customers require recurring services over a multi-period planning horizon. The usual tasks of these problems are to assign customers to each vehicle and determine vehicle routes simultaneously in such a way that the cost of routing is minimal. Also, these problems do not emphasise the compactness of a cluster of customers: see Francis et al. (2008) and Irnich et al. (2014) for more details. Therefore, they do not apply to our problem.

The study by Mourgaya & Vanderbeck (2007) is different to those works in the periodic vehicle routing problems. As well as assigning schedules to customers, the authors allocated customers to a vehicle in such a way that a cluster of customers for each vehicle is balanced and compact. Usually, the optimised route is determined in the periodic vehicle routing problem but the authors suggested deciding it at a later stage. The objective function is to maximise the compactness of clusters while keeping them balanced in terms of workload. Nevertheless, the planning horizon in this work is only up to six days which is not enough to consider week clusters. In fact, this problem does not include a plan to serve customers at both week and day level like the MPSDP.

To the best of our knowledge, Bender et al. (2016) were the first to address the aspect that customers require recurring services with different frequencies in the sales districting problem. In their study, the customers can request more than one visit in each visiting week and have a specific preference on a visiting day(s). Moreover, customers might require different service time in each visit so their service time can be varied during the planning horizon. Note that the information regarding service times for every customer is known in advance.

The multi-period sales districting problem is extremely challenging to solve optimally for large scale instances using a standard linear solver (Bender et al. 2016). Moreover, districts and schedules for salesmen are required to be updated on different time scales. Districts for salesmen run for several years, while schedules are usually revised more frequently, for example, every few months. Therefore, Bender et al. (2016) suggested splitting the problem into two main parts. The first one is the *districting part*, which is similar to the classical sales districting problem: customers are partitioned into districts with the satisfactory attributes for each salesman. In each district, the authors then separately considered the *scheduling part*. For the predefined set of customers, this part attempts to design high-quality schedules to serve the customers.

In particular, the authors determined schedules that satisfy customers' frequency requirements and have desirable attributes; compactness, contiguity and workload balance. Now, when only schedules are required to be revised, the scheduling part is solved where the set of customers in each district is unchanged. Whenever districts are required to update, the authors suggested solving both parts sequentially, i.e., first the districting part and then the scheduling part.

As the study of compactness in schedules is still rare in the literature, they focused on solving only the scheduling part for a given compact and balanced district. They proposed a mixed-integer programming formulation for the scheduling part. In this case, they aimed for weekly schedules and daily schedules that satisfy customers' visiting requirement in both levels. Both schedules are required to be compact and balanced as these two attributes provide benefit to the performance of salesmen.

The main tasks of the scheduling part are to determine visiting weeks and also visiting days in those weeks for each customer. At the same time, the centre in each week cluster and daily cluster, which can be any customer in the district, has to be decided for measuring compactness. The weekly and daily compactness are calculated from the total centre-based distance in every week cluster and every daily cluster, respectively. The objective function, in this case, is to maximise the convex combination between the weekly compactness and the daily compactness. At the same time, the workloads on the week level and on the day level have to be within the acceptable ranges.

According to preliminary tests in Bender et al. (2016), even the scheduling part of the MPSDP is difficult to solve for small data instances, e.g., $30-50$ customers with four weeks and five days per week. A modern linear solver struggles to reach even a small optimality gap within an hour in the tests, due to a large number of variables. More importantly, solutions for the scheduling part are highly symmetrical, i.e., it is possible to rearrange week clusters in different ways without changing the objective function value: see an example in Bender et al. (2016). Therefore, the scheduling part requires more sophisticated approaches to derive high-quality solutions as we will discuss in the next subsection.

2.2.2 Solution Approaches

The scheduling part of the MPSDP has been studied only recently and it is challenging to solve even for small data instances. To the best of our knowledge, there are only two solution approaches proposed so far.

First, Bender et al. (2016) implemented a location-allocation heuristic. To the best of our knowledge, this was the first extension of Hess et al. (1965) to apply to a multi-period framework.

Initial week and day centres are required to start the location-allocation heuristic. Usually, in the case of a single-period districting problem, initial centres are selected in such a way that they distribute evenly across an area of study to avoid overlapping initial districts. Unfortunately, any standard method used to create initial centres in a single-period case is not applicable to the MPSDP since week patterns and weekday patterns of customers must be taken into account. Bender et al. (2016) adapted the k-means++ algorithm (Arthur & Vassilvitskii 2007) to derive initial week and day centres. In particular, the authors favoured customers who tend to be good centres. First, customers who have a smaller week rhythm should have more chance to be selected as initial week or day centres since they tend to be presented in most of the week clusters. Additionally, at each level, customers who are close to already selected centres should be avoided as they increase the risk of overlapping districts. Moreover, day centres should be determined in such a way that they are close to their week centre.

The algorithm was tested on real-world data instances which are related to business that requires salesmen to visit retailers regularly, for example, supermarkets and gas stations. The number of customers in the instances was 115 on average. The planning horizon was 16 weeks where each week consists of 5 days. Bender et al. (2016) evaluated the solution in terms of compactness, approximate total travel time, and balance of schedules. Namely, the compactness in the evaluation was measured from the average distance between any two customers who are in the same week or day cluster. The authors approximated the total travel time by the summation of daily routes. Here, the daily routes were derived by solving a symmetric travelling salesman problem (TSP), where every salesman starts and ends their routes at their offices. They also

checked the total approximate travel time inside day clusters to gain more insights. Finally, the weekly and daily balance of schedules were derived from the maximum relative deviation from the average weekly and daily service time, respectively.

The computational results showed that the method generates high-quality solutions quickly within 5 minutes which is an acceptable amount of time for real practitioners. They also studied the effect on travel time by the weekday regularity, i.e., when customers have specific weekday patterns. More precisely, they compared solutions where customers strictly require the same weekday pattern in every visiting week (called strict weekday regularity) to those solutions that allow deviation from the regular weekday patterns (called partial weekday regularity). The results confirm that the weekday regularity increases travel time in general. However, it is inconclusive if the strict or the partial weekday regularity tend to increase more amount of travel time. This is because the service time and the week rhythms are key factors on how much the travel time increases.

Bender et al. (2018) proposed another solution method for the scheduling part of the MPSDP, in particular, they aimed to design weekly and daily schedules for customers who require recurring services. However, there are a few differences in the problem details. Here, customers require only one service per visiting week. Also, customers do not have any specific requirement on visiting days, i.e., the weekday regularity is not considered in this case. Moreover, the service time of every customer is fixed during the planning horizon. Although the problem is more specific than the one in Bender et al. (2016), Bender et al. (2018) assured that this problem is still highly relevant. The authors proposed a branch-and-price technique as an exact method to solve the scheduling part. The main novelty in implementation here is an additional feature to reduce the symmetry in a solution during the branching process. Numerical experiments were conducted on real-world small data instances with at most 55 customers on a planning horizon of 4 weeks and 5 days per week. The instances were based on data from a German manufacturer of paints and coatings. The results showed this new feature reduces the computational time by 98.1% compared to a standard linear solver.

As we can see, there are only a few approaches for solving the scheduling part of the MPSDP. Moreover, any single integrated method for solving the MPSDP has not been proposed yet. Although Bender et al. (2016) suggested solving the scheduling part and the districting part sequentially for the MPSDP, this approach is at risk of providing sub-optimal solutions. Therefore, we aim to fulfil the need for more approaches to the MPSDP. Our contribution will be presented in Section 2.3.

2.3 Contributions of the Research

In our research, we extend the work of Bender et al. (2018). Namely, we assume that each customer requires one visit per visiting week and does not have any restriction on visiting day. According to Bender et al. (2016), a fifth of customers change their visiting day in the short term. Since now every customer can be served any day during a week in this case, we believe that it is more efficient to plan daily schedules later, e.g., nearer the operation time. Therefore, we emphasise only weekly schedules and do not focus on designing daily schedules here.

Our contributions are as follows:

1. We formally present a model formulation for the multi-period sales districting problem, i.e., a formulation that combines the districting part and the scheduling part.
2. We propose more elaborate solution approaches, including an exact method and a heuristic, to solve the scheduling part. Here, we develop Benders' decomposition as an exact solution method, and tabu search as a heuristic by using various sophisticated techniques. We perform the computational experiments to evaluate their effectiveness and efficiency on randomly generated data instances with $30-50$ customers and a maximum of 15 weeks.
3. For the first time, we develop a sophisticated technique that solves the districting part and the scheduling part simultaneously. In particular, we extend the tabu search that solves the scheduling part to solve the whole problem. Numerical results on large randomly generated data instances with $100-300$ customers and a maximum of 15 weeks are presented. To support the robustness of our algorithm, we also show that a modern linear solver like CPLEX struggles to solve these large data instances, while our algorithm is

capable of providing high-quality solutions within a reasonable amount of time.

2.4 Mathematical Formulations

2.4.1 Scheduling Part of the Problem

A mathematical formulation for the scheduling part of the problem was, for the first time, proposed by Bender et al. (2016). As we consider only weekly schedules and assume that each customer requires only one visit per visiting week, we simplify the original model as follows.

Let B be the set of customers and $W = \{1, ..., |W|\}$ be the set of weeks in a planning horizon, where $|W|$ is the total number of weeks. P_b defines the set of possible week patterns for customer $b \in B$, and $P = \bigcup_{b \in B} P_b$ is the set of all possible week patterns. A parameter ψ_p^w, $p \in P, w \in W$, defines a week in a week pattern, as shown below.

$$\psi_p^w = \begin{cases} 1, & \text{if week pattern } p \in P \text{ includes week } w \in W, \\ 0, & \text{otherwise.} \end{cases}$$

In this model, each customer $b \in B$ has the specific week rhythm $r_b \in \mathbb{N}^+$, i.e., they require service every r_b weeks. As a result, $|W|$ should be the least common multiple of $\{r_b\}_{b \in B}$ to guarantee that weekly schedules can be repeated. We assume that all customers demand one visit per visiting week. Therefore, there are $\frac{|W|}{r_b}$ visits for customer b during the planning horizon.

Let t_b be the constant service time for customer $b \in B$. The average weekly service time is denoted by $\mu^{week} = \frac{TWL}{|W|}$, where $TWL = \sum_{b \in B} t_b \cdot \frac{|W|}{r_b}$ is the total workload during the planning horizon. τ^{week} represents a predefined maximum deviation from the average workload, which is used to define the acceptable ranges of workload each week.

Regarding decision variables for this model, the ones to choose a week pattern for each customer are defined as

$$g_{bp} = \begin{cases} 1, & \text{if week pattern } p \in P \text{ is selected for customer } b \in B, \\ 0, & \text{otherwise.} \end{cases}$$

Since we approximate compactness by a centre-based measurement, we must define more decision variables to choose week centres and allocate the associated customers to the centre of each week. Therefore, more decision variables are presented in the following way:

$$x_b^w = \begin{cases} 1, & \text{if customer } b \in B \text{ is the week centre in week } w \in W, \\ 0, & \text{otherwise;} \end{cases}$$

$$u_{bi}^w = \begin{cases} 1, & \text{if customer } b \in B \text{ is assigned to week centre } i \in B \text{ in week } w \in W, \\ 0, & \text{otherwise.} \end{cases}$$

The objective function of the model is to maximise the compactness by minimising the total centre-based distance incurred in the planning horizon, where c_{bi} represents the distance between customers b and i, for $i, b \in B$.

Our formulation for the scheduling part of the multi-period sales districting problem is

shown below.

$$\min \quad \sum_{b\in B}\sum_{i\in B}\sum_{w\in W} c_{bi}u_{bi}^{w} \qquad \text{(OriginalSchedule)}$$

$$\text{s.t.} \quad \sum_{p\in P_b} g_{bp} = 1 \qquad b\in B \qquad (2.1)$$

$$\sum_{i\in B} u_{bi}^{w} = \sum_{p\in P_b} \psi_p^w g_{bp} \qquad b\in B,\ w\in W \qquad (2.2)$$

$$u_{bi}^{w} \le x_i^w \qquad b,\ i\in B,\ w\in W \qquad (2.3)$$

$$\sum_{b\in B} x_b^w = 1 \qquad w\in W \qquad (2.4)$$

$$\sum_{b\in B, p\in P_b} t_b\psi_p^w g_{bp} \ge (1-\tau^{week})\mu^{week} \qquad w\in W \qquad (2.5)$$

$$\sum_{b\in B, p\in P_b} t_b\psi_p^w g_{bp} \le (1+\tau^{week})\mu^{week} \qquad w\in W \qquad (2.6)$$

$$g_{bp}\in\{0,1\} \qquad b\in B,\ p\in P_b \qquad (2.7)$$

$$x_b^w\in\{0,1\} \qquad b\in B,\ w\in W \qquad (2.8)$$

$$u_{bi}^{w}\ge 0 \qquad b,\ i\in B,\ w\in W. \qquad (2.9)$$

The objective function of (OriginalSchedule) is to minimise the total distances between customers and their week centres in every weekly district during the planning horizon. Constraints (2.1) guarantee one week pattern for each customer. Constraints (2.2) and (2.3) ensure that if a customer requires service in a specific week, they must be assigned to the week centre of that week. Constraints (2.4) ensure a unique week centre in every week. Constraints (2.5) and (2.6) define the acceptable ranges of total service times in each week. These are to guarantee the workload balance of every week cluster. Finally, Constraints (2.7)−(2.9) are the domain constraints for the decision variables. Although the variables u_{bi}^{w} are not defined as binary, they will be binary due to the objective function, and Constraints (2.2) and (2.3).

2.4.2 The MPSDP

In the scheduling part, we assume that we already have a compact and balanced district to determine desirable weekly schedules. Here, we assume that we have a set of customers and a set of salesmen. An additional task from the scheduling part is to allocate customers to each salesman so that every district is compact and workload balanced. In other words, this time, we aim to construct balanced and compact districts, and, at the same time, desirable weekly schedules for salesmen.

Let $D=\{1,\dots,|D|\}$ be the set of districts for salesmen, where $|D|$ is the number of the districts. $B=\{1,\dots,|B|\}$ is the set of customers, where $|B|$ is the total number of customers.

To create districts, we introduce new binary variables to allocate customers to each district, v_{bd}, $b\in B, d\in D$, which are defined in the following way:

$$v_{bd}=\begin{cases}1, & \text{if customer } b\in B \text{ is assigned to district } d\in D,\\ 0, & \text{otherwise.}\end{cases}$$

For the information on the customers such as their locations and week rhythms, we use the same notation as in Section 2.4.1. Let t_b and r_b be the constant service time and the week rhythm of customer $b\in B$. Assuming that all customers require one visit in each visiting week, the total service time along the planning horizon is $TWL=\sum_{b\in B} t_b\cdot\frac{|W|}{r_b}$. The average workload for each district is represented by $\mu^{dis}=\frac{TWL}{|D|}$. The maximum acceptable deviation from the average workload in each district is controlled by a parameter τ^{dis}.

Concerning the weekly workload balance constraints, the average workload each week in every district is $\mu^{week}=\frac{\mu^{dis}}{|W|}$ and τ^{week} is the acceptable deviation from the weekly average.

Regarding weekly schedules for each district, all notation is similar to that used in Section 2.4.1. Let $W = \{1, \ldots, |W|\}$ be the set of weeks in schedules, where $|W|$ is the planning horizon. P_b defines the set of possible week patterns for customer $b \in B$. $P = \bigcup_{b \in B} P_b$ is the set of all possible week patterns in the problem. A parameter ψ_p^w, $p \in P, w \in W$, denotes a week in a week pattern, i.e.,

$$\psi_p^w = \begin{cases} 1, & \text{if week pattern } p \in P \text{ has week } w \in W, \\ 0, & \text{otherwise.} \end{cases}$$

Decisions on the week pattern for each customer in a district, the week centre in each week, and the customer allocation to the week centres have to be made, similarly to the scheduling part from Section 2.4.1. We use the same notation of variables for those decisions but introduce another dimension: the set of districts D.

$$g_{bpd} = \begin{cases} 1, & \text{if week pattern } p \in P \text{ is for customer } b \in B \text{ in district } d \in D, \\ 0, & \text{otherwise;} \end{cases}$$

$$x_{bd}^w = \begin{cases} 1, & \text{if customer } b \in B \text{ is the week centre in } w \in W \text{ of district } d \in D \text{ ,} \\ 0, & \text{otherwise;} \end{cases}$$

$$u_{ibd}^w = \begin{cases} 1, & \text{if customer } b \in B \text{ in district } d \in D \text{ is assigned to week centre } i \in B \text{ in week } w \in W, \\ 0, & \text{otherwise.} \end{cases}$$

We still use a centre-based measurement to qualify the compactness on the district level and on the week level. On the district level, the district centres have been predefined already. In this case, they are offices of salesmen. On the week level, the centres of week clusters are determined during the optimisation process. The compactness on the district level is the total distance between customers and their district centres, while the compactness on the week level is quantified by the total distance between customers and their week centres. Let $\overline{c}_{bd}$ be the distance between customer $b \in B$ and the centre of district $d \in D$. The distance between customer i and b, $i, b \in B$ is represented by c_{ib}. The objective function for the problem is to minimise the convex combination between the compactness on the district level and on the week level, where λ is a weight on the compactness in the district level.

A mathematical model formulation for the MPSDP is shown below.

$$\min \quad \lambda \sum_{d\in D}\sum_{b\in B} \bar{c}_{bd} v_{bd} + (1-\lambda) \sum_{d\in D}\sum_{b\in B}\sum_{i\in B}\sum_{w\in W} c_{ib} u^w_{ibd} \qquad \text{(Model\_MPSDP)}$$

$$\text{s.t.} \quad \sum_{d\in D} v_{bd} = 1 \qquad b \in B \qquad (2.10)$$

$$\sum_{d\in D}\sum_{p\in P_b} g_{bpd} = 1 \qquad b \in B \qquad (2.11)$$

$$g_{bpd} \le v_{bd} \qquad b \in B,\ p \in P_b,\ d \in D \qquad (2.12)$$

$$u^w_{ibd} \le x^w_{id} \qquad b,\ i \in B,\ w \in W,\ d \in D \qquad (2.13)$$

$$\sum_{i\in B} u^w_{ibd} = \sum_{p\in P_b} \psi^w_p g_{bpd} \qquad b \in B,\ w \in W,\ d \in D \qquad (2.14)$$

$$\sum_{b\in B} x^w_{bd} = 1 \qquad w \in W,\ d \in D \qquad (2.15)$$

$$\sum_{b\in B} \frac{t_b \cdot |W| \cdot v_{bd}}{r_b} \ge (1-\tau^{dis})\mu^{dis} \qquad d \in D \qquad (2.16)$$

$$\sum_{b\in B} \frac{t_b \cdot |W| \cdot v_{bd}}{r_b} \le (1+\tau^{dis})\mu^{dis} \qquad d \in D \qquad (2.17)$$

$$\sum_{b\in B, p\in P_b} t^w_b \psi^w_p g_{bpd} \ge (1-\tau^{week})\mu^{week} \qquad w \in W,\ d \in D \qquad (2.18)$$

$$\sum_{b\in B, p\in P_b} t^w_b \psi^w_p g_{bpd} \le (1+\tau^{week})\mu^{week} \qquad w \in W,\ d \in D \qquad (2.19)$$

$$g_{bpd} \in \{0,1\} \qquad b \in B,\ p \in P_b,\ d \in D \qquad (2.20)$$

$$v_{bd} \in \{0,1\} \qquad b \in B,\ d \in D \qquad (2.21)$$

$$u^w_{ibd} \ge 0 \qquad b,\ i \in B,\ d \in D,\ w \in W \qquad (2.22)$$

$$x^w_{bd} \in \{0,1\} \qquad b \in B,\ d \in D, w \in W. \qquad (2.23)$$

The objective function here is to maximise the convex combination of the compactness on the district level and on the week level. Constraints (2.10) ensure that every customer is assigned to a unique district. Constraints (2.11) show that only one week pattern is assigned for each customer. Constraints (2.12) guarantee that a customer can have a week pattern in a district only when they are assigned to the district. Constraints (2.13) and (2.14) ensure that if a customer requires service in any specific week, they must be assigned to the week centre of the corresponding week cluster. Constraints (2.15) limit one week centre in each week cluster for every district. Constraints (2.16)−(2.19) define the acceptable workload in each district and each week cluster. Finally, Constraints (2.20)−(2.23) are the domain of decision variables. Although u^w_{ibd}'s are continuous variables, they are binary variables under the influence of Constraints (2.13) and (2.14), similarly to (OriginalSchedule).

2.5 Experiment on Small Data Instances

As we discussed in the literature review, the scheduling part of the MPSDP is already hard to solve. To emphasise the complexity of the problem, in this section we will show that CPLEX 12.7.1 which is a state-of-the-art linear solver struggles to solve the scheduling part even on small data instances.

All of the results are derived using a computer with an Intel Core i5-8365U processor and 16 GB of RAM under a 64-bit-Windows 10 operating system. All code is written in Java in Eclipse IDE for Java Developers using CPLEX 12.7.1 to solve a mixed-integer programme.

We generate random data instances, containing the information of customers and their week rhythms, for our experiment as follows. First, we generate the instances with a group of 30, 40 and 50 customers. For each group of customers, there are three sets of possible week rhythms

Data	Best %Gap	%Gap	Worst %Gap	#Opt (45)	Total time (s)
30_1	0.00	0.01	0.01	5	15
30_2	0.00	0.01	0.01	5	120
30_3	2.57	7.81	23.54	0	900
40_1	0.00	0.01	0.01	5	30.8
40_2	0.01	0.01	0.01	5	291.2
40_3	14.60	23.34	28.53	0	900
50_1	0.01	0.01	0.01	5	50.8
50_2	3.47	4.57	6.21	0	900
50_3	23.20	32.10	41.33	0	900
Total				25	

Table 2.1: The performance of CPLEX by the default setting.

with a different planning horizon. Let f_i be the week rhythm of customer i in a data instance. The sets of the possible week rhythms are:

1. $f_i \in \{1, 2, 4, 8\}$ in a total of eight weeks, i.e., customers demanding service from a salesperson every week, two weeks, 4 weeks or 8 weeks in the total planning horizon time of 8 weeks.
2. $f_i \in \{1, 2, 3, 4\}$ in a total of 12 weeks.
3. $f_i \in \{2, 3, 5\}$ in a total of 15 weeks.

We have nine combinations of the number of customers and the set of possible week rhythms. Throughout the current and subsequent chapters, the name of each combination is denoted by *the number of customers_the set of week rhythms*. For example, 30_3 defines the combination which contains 30 customers, and each of them can have a week rhythm of two weeks, three weeks or five weeks in a 15-week plan. For each combination, we generate 5 data instances randomly so there are 45 instances in total in the experiment. Note that the more customers and the longer the planning horizon, the more challenging it becomes to solve for optimality due to the increasing number of integer variables.

For each instance, the week rhythm for each customer is drawn from a discrete uniform distribution. The constant service time for each customer in each visiting week follows a discrete uniform distribution between 15 and 80 minutes. The location of each customer is represented by a point in the two-dimensional Cartesian coordinate system where each coordinate, independently, follows a continuous uniform distribution between 0 and 10. The distance between customers is squared Euclidean distance. The maximum deviation from the average weekly workload (τ^{week}) is 0.05.

The experiment is run on a single thread. The satisfactory tolerance level between the bounds in the tree is 0.01%, which is the default value for CPLEX. The maximum computational time for each data instance is 900 seconds (15 minutes).

Table 2.1 compares the results in terms of the relative percentage gaps between the upper bound and the lower bound of the objective value, and the average total execution time in seconds. For each combination, we present the best, the average, and the worst relative percentage gaps among five instances in *Best %Gap*, *%Gap*, and *Worse %Gap*, respectively. Note that the best and the worst value are, respectively, the minimum and the maximum relative percentage gap among these five data instances. Further information that relates to the relative percentage gaps is the total number of instances in each combination that find the optimal solution, i.e., those whose relative percentage gaps are less than 0.01%. This information is presented in Column *#Opt*, where the number in brackets is the maximum total number of *#Opt*, which, in this case, is the total number of instances in the experiment. The last row, *Total*, presents the total of *#Opt* from every combination for each algorithm to show the overall results. The last column, *Total time (s)*, shows the average computational time in seconds.

For the less complicated combinations, e.g., 30_1, 30_2, 40_1, 40_2, and 50_1, CPLEX manages to find the optimal solution in every instance within the reasonable computational time on

average, i.e., less than 5 minutes. However, for the rest of the more complicated combinations, CPLEX cannot find the optimal solution under the limitation of time. In the worst cases, the relative percentage gaps can reach between 23% and 42%, which are noticeably high. Overall, CPLEX can find the optimal solutions in only around 55% of the total instances.

Obviously, the modern linear solver cannot always solve the scheduling part to optimality even for the small number of customers. Therefore, we propose sophisticated solution approaches to derive efficient schedules in the next chapters.

Chapter 3

Benders' Decomposition for the Scheduling Part

Benders (1962) first proposed Benders' decomposition for solving mixed-variables programming problems. The method is beneficial for problems that become simpler after the values of complicating variables (the variables that make the problems more difficult to solve) are temporarily fixed. It is also suitable for problems where the constraint matrix exhibits a partitioning or block structure. Benders (1962) mainly focused on a problem that reduces to a linear programme when the complicating variables (for example, integer variables in a mixed-integer problem) are fixed. Here, we are interested in the same kind of problems. For more general cases, the interested reader is referred to Geoffrion (1972).

According to Geoffrion (1970*a*) and Geoffrion (1970*b*), Benders' decomposition contains a process of projection, outer linearisation, and relaxation. It begins with projecting the feasible region of the problem onto the subspace of the complicating variables. For fixed values of the complicating variables, the problem thus reduces to an ordinary linear programme; it is then dualised and used to generate optimality cuts and feasibility cuts from its extreme points and extreme rays, respectively. Note that the outer linearisation is used to define the optimality cuts from the extreme points. The optimality cuts provide the information of the projected costs, while the feasibility cuts are the feasibility requirements for the complicating variables (Rahmaniani et al. 2017). With these processes, we can derive a reformulation that contains the complicating variables, their associated constraints, and the optimality cuts and the feasibility cuts resulting from the projection. Since the total number of optimality cuts and feasibility cuts is usually huge, the reformulation is often intractable, or even impossible to solve. Therefore, we relax the optimality cuts and the feasibility cuts, and then add only those which are necessary to find the optimal solution.

Namely, in the beginning, the relaxation of the reformulation, called the *master problem*, has only constraints for the complicating variables and does not have any optimality cuts and feasibility cuts. The master problem is solved for the complicating variables. Then, it provides the complicating variable values to a *subproblem*. With the fixed complicating variables, the subproblem generates an optimality cut or a feasibility cut that has been violated. That cut is added to the master problem to guide further processes. The method repeats these steps until it reaches a stopping criterion.

Since Benders' decomposition can exploit the structure of problems to reduce the computational burden, it has gained a lot of attention, especially in this decade, when tackling various types of optimisation problems. According to Rahmaniani et al. (2017), the method is popular not only in linear mixed-integer programming (Contreras et al. 2012, de Sá et al. 2013, Taşkın & Cevik 2013, Gelareh et al. 2015, Kergosien et al. 2017, Fontaine & Minner 2018), but also in stochastic problems (Santoso et al. 2005, Contreras et al. 2011*b*, Adulyasak et al. 2015, Boland et al. 2016). Moreover, it is successful in tackling nonlinear problems: see de Camargo et al. (2011); Gendron et al. (2013); Mariel & Minner (2017); Tapia-Ubeda et al. (2018).

Regarding applications, to the best of our knowledge, the method has never been applied to districting problems. However, it shows success in closely related applications like facility

location (Tang et al. 2013, Naoum-Sawaya & Elhedhli 2013, Vatsa & Jayaswal 2016, Fischetti et al. 2017, Pearce & Forbes 2018); network design (Cordeau et al. 2006, Santibanez-Gonzalez & Diabat 2013, Pishvaee et al. 2014, Gendron et al. 2016, Rahmaniani et al. 2018); and hub location problems (de Camargo et al. 2009, Üster & Agrahari 2011, Contreras et al. 2011*a*, de Sá et al. 2018). Therefore, the method is promising to try on our model.

According to Rahmaniani et al. (2017), the original Benders' decomposition often struggles to solve problems with instances of practical size due to the following limitations. First, after the projection and the relaxation process in the decomposition, the master problem loses some information about the original problem to the subproblem. As a result, the master problem usually has a weak relaxation, especially in the first few iterations, which directly affects the quality of a cut derived from its solution. Also, solutions of the master problem might show an oscillation behaviour when they approach the optimal solution. Namely, the current master solution can move significantly far away from those of previous iterations. These effects increase with the number of iterations and slow down the convergence of the algorithm. Time consumption in each iteration is also a main bottleneck of the algorithm. This can stem from solving the time-consuming master problem to optimality, or poor-quality cuts generated from the subproblem. Therefore, many acceleration techniques for the master problem and the subproblem to increase the algorithm's efficiency have been proposed. Here we provide brief details on some of those methods. More thorough information of enhancements to Benders' decomposition can be found in Rahmaniani et al. (2017).

Tightening the relaxation of the master problem, in the beginning, speeds up the algorithm. In the first iterations, the quality of the solutions is often poor due to a lack of information from the subproblem. Warm-start strategies are usually used to tackle this problem. One of the common strategies is using heuristics to generate a set of initial high-quality feasible solutions. Those solutions are used not only for deriving initial cuts to tighten the relaxed master problem, but also for providing a good upper bound in the case of minimisation problems (Contreras et al. 2011*a*). Another conventional approach is adding valid inequalities to the master problem. Some valid inequalities aim to improve an initial lower bound of the master problem: see, e.g., Adulyasak et al. (2015). Some valid inequalities are for avoiding infeasible solutions, thereby excluding feasibility cuts from the master problem, since feasibility cuts do not improve a bound of a problem (Pishvaee et al. 2014, Mariel & Minner 2017).

Strengthening the relaxation of the master problem is the main key to enhance the algorithm. For mixed-integer master problems, a popular approach is the two-phase Benders' decomposition by McDaniel & Devine (1977). The authors showed that Benders' cuts generated from solutions of the linear programming (LP) relaxation of the master problem are valid for a problem. They exploited this fact in the first phase; they considered the LP relaxation of the master problem and generated cuts from fractional solutions. Then, in the second phase, they reintroduced the integrality conditions back to the master problem and implemented the classical Benders' decomposition. This technique tightens the LP relaxation of the master problem and shows success in many studies, e.g., Papadakos (2009), Botton et al. (2013), Adulyasak et al. (2015). When the subproblem has a decomposable structure, i.e., it contains smaller independent problems, generating each cut from each independent problem can also strengthen the relaxation of the master problem. Successful results have been reported in many papers, e.g., Pishvaee et al. (2014), Mariel & Minner (2017), Fontaine & Minner (2018), Pearce & Forbes (2018). However, this strategy directly increases the size of the master problem. If there are too many cuts, the master problem will have more computational burden instead. Therefore, it needs careful implementation. To exploit this strategy, Birge & Louveaux (1997) suggested that the total number of cuts from these smaller subproblems in each iteration should not be much larger than the total number of constraints of the master problem without the Benders' cuts. Contreras et al. (2011*a*) showed how to create multiple cuts properly for an uncapacitated hub location for a multi-commodities problem. With a predetermined decision on opening hubs from the master problem, the subproblem is a transportation problem that can be separated either for each commodity or for each hub node. The total number of commodities can be much larger than the total number of hubs for each instance in the study. Therefore, the authors decomposed the subproblem for each hub node to avoid too many cuts per iteration.

A procedure to solve the master problem also affects the computational time of each iteration. In case the master problem is an integer programme, solving the integer programme

iteratively to optimality is a main computational burden (Magnanti & Wong 1981). Geoffrion & Graves (1974) suggested a way to reduce the computational time by not solving the master problem to optimality, especially in the first iterations. More significantly, an effective and modern technique is the branch-and-Benders' cut. This strategy builds a single search tree for the master problem and generates Benders' cuts from each node (either with fractional solutions or integer solutions) in the tree. This can save a significant amount of time since we do not create a new branch-and-bound tree for the integer programme and solve it from scratch in every iteration. Also, nowadays, state-of-the-art optimisation solvers have accessible features to intervene in the branch-and-bound tree and facilitate a manual design on generating cuts inside the tree. Therefore, this method has gained much more attention in this decade, especially for tackling problems with large-sized instances (for example, Contreras et al. (2012), de Sá et al. (2013), Botton et al. (2013), Fischetti et al. (2017)). A decision about when to generate cuts in the tree is also important because it directly affects the size of the tree. It can vary from adding cuts at every feasible node (with either a fractional solution or an integer solution) to adding cuts only at nodes with integer solutions. Botton et al. (2013) do not recommend generating cuts in every node. They suggested adding as many Benders' cuts from fractional solutions as possible only at the root node to tighten the LP relaxation of the master problem, similarly to the first phase of two-phase Benders' decomposition. After branching the root node, it is sufficient to add cuts only at a node with an integer solution. Many studies confirmed that this cut strategy is effective in tackling difficult problems: see Fortz & Poss (2009), de Camargo et al. (2011), Adulyasak et al. (2015), de Sá et al. (2018), Pearce & Forbes (2018).

Regarding instability on solutions of the master problem, regularised decomposition, trust-region and level decomposition are regular techniques to overcome the issue. These techniques, however, add complexity to the master problem and they are not practical for combinatorial optimisation problems (Rahmaniani et al. 2017). Therefore, we do not focus on them here.

The quality of Benders' cuts is another essential factor for the effectiveness of the algorithm. When the subproblem is highly degenerate, there can be multiple dual solutions that produce Benders' cuts of different strengths. Magnanti & Wong (1981) defined a Pareto-optimal cut as a nondominated optimality cut. For a given optimal objective value of the dual subproblem, the authors proposed an auxiliary subproblem to create the strongest cut from a dual subproblem solution with the same optimal objective value. The auxiliary problem utilised a core point which is a point in the relative interior of the convex hull of the master problem. Papadakos (2008) addressed the weakness of the idea from Magnanti & Wong (1981) as the auxiliary problem tends to have a numerical instability issue. Also, core points are usually not easy to find. Papadakos (2008) proposed a way to remove the instability issue by modifying the auxiliary problem to be independent of the optimal objective value of the dual subproblem. Also, the author showed how to update a new core point for every iteration: it is a convex combination of a previously used core point and a solution of the master problem. This way requires only an initial valid core point to start. However, this does not apply when a master solution yields an infeasible primal subproblem. de Sá et al. (2013) extended the work of Papadakos (2008) for such a case; they adjusted the weight on the convex combination so that the updated core point is still valid to generate a Pareto-optimal cut. Naoum-Sawaya & Elhedhli (2013) contributed another way to generate Pareto-optimal cuts. They substituted a core point with an analytic centre and proved that it generates a Pareto-optimal cut. The approach showed promising results in solving the capacitated facility location and multi-commodity capacitated fixed-charge network design problems. Nevertheless, the success of the algorithm mainly relies on the quality of the core points and the efficiency of the re-optimisation methods.

Deriving core points is usually the main challenge in the process of generating Pareto-optimal cuts because they are not easy to find for some complicated problems (see, for example, Mercier et al. (2005)). For those problems, approximate core points are always used instead. For example, Santoso et al. (2005) used a solution from the linear relaxation of the master problem; Papadakos (2008) used a master solution from the first iteration as a substitution for the initial core point. However, these methods do not guarantee the generation of Pareto-optimal cuts and might not improve the efficiency of the algorithm. Sherali & Lunday (2013) addressed this issue by introducing another kind of a nondominated optimality cut called maximal nondominated. This cut generation is less strict than the Pareto-optimal cut from Magnanti & Wong (1981). Its main advantage is that it requires only a positive weight vector instead, thereby avoiding

the challenges related to generating valid core points.

Moreover, the algorithms proposed in Magnanti & Wong (1981) and Papadakos (2008) do not always provide a net computational benefit due to having to solve two linear programmes in every iteration, i.e., the dual subproblem and the auxiliary dual subproblem (Mercier & Soumis 2007). Therefore, in the process of generating the maximal nondominated cuts, Sherali & Lunday (2013) combined these two linear programmes into one problem under a multi-objective optimisation framework. In particular, the objective function of the model reformulation is a weighted sum of the objective functions of these two problems. In experiments on instances of the fixed-charge network problem, the method showed superior results to the benchmark, which was using the Pareto-optimal cuts by Magnanti & Wong (1981) with the core point approximation method from Papadakos (2008). However, the technique still has one main issue: a proper value of the weight parameter in the objective function is not practical to derive. Oliveira et al. (2014) then suggested dynamically adjusting the value of the weight parameter so that it mostly focuses on improving the original objective function of the dual subproblem at a later stage of the algorithm. This is to ensure the convergence of the algorithm to be the same as the classical Benders' decomposition scheme. Experiments on instances of two-stage stochastic programming supported the robustness of the algorithm which was superior to those of Magnanti & Wong (1981) and Sherali & Lunday (2013).

Typically, Benders' cuts are generated separately according to the subproblem status; an optimality cut from a subproblem which is solved to optimality and a feasibility cut from an infeasible subproblem. Fischetti et al. (2010) proposed a different way to generate Benders' cuts. They reformulated the subproblem as a pure feasibility problem that can generate both optimality cuts and feasibility cuts in the same framework. The computational results on instances of the multi-commodity network flow problems supported the robustness of the algorithm.

This chapter provides the theoretical background of Benders' reformulation on a mixed-integer linear programme in Section 3.1. Then, the classical Benders' decomposition is presented in Section 3.2. We show how to apply the method to our model in Section 3.3. However, the technique usually struggles to deal with instances whose size is practically relevant. Therefore, Section 3.4 presents techniques to improve the efficiency of the method. These include the two-phase Benders' decomposition, a heuristic for warm-start, nondominated optimality cuts (either Pareto-optimal cuts or maximal nondominated cuts), multiple cuts from the decomposable subproblem, and the modern branch-and-Benders' cut. Section 3.5 provides the details of the programme set-up for our experiment, followed by the discussion about the effectiveness of every technique and the comparison of results between our developed method and the Benders' decomposition algorithm of CPLEX in Section 3.6.

3.1 Benders' Reformulation

In this section, we show how to derive Benders' reformulation for a mixed-integer linear programme. Here, we assume that the complicating variables are integer variables. Once the integer variables are fixed, the problem is reduced to a linear programme, and a standard duality theory is exploited to derive the reformulation.

A mixed integer linear programme can be represented as follows:

$$\begin{aligned} \min \quad & f^T y + c^T x \\ \text{s.t.} \quad & Ay = b \\ & By + Dx = d \\ & x \geq 0 \\ & y \geq 0 \quad \text{and integer.} \end{aligned} \tag{OriginalBD}$$

Suppose that $\overline{y}$ satisfying $A\overline{y} = b$ are the fixed integer variables. Then, we get the resulting linear programme as the following:

$$\min_{x \geq 0} \{c^T x \mid Dx = d - B\overline{y}\}. \tag{SPBD($\overline{y}$)}$$

Since (SPBD($\overline{y}$)) is a linear programming problem, we can consider its dual problem instead. Let π be the dual variables corresponding to the constraints $Dx = d - B\overline{y}$. The corresponding dual problem is

$$z_{LP}(\overline{y}) := \max\{\pi^T(d - B\overline{y}) \mid \pi^T D \leq c^T,\ \pi \text{ is free}\}. \qquad \text{(dualSPBD}(\overline{y})\text{)}$$

Next, we can represent the polyhedron of (dualSPBD($\overline{y}$)) in terms of its extreme points and extreme rays.

Let R be the feasible region of (OriginalBD) where

$$R = \{(x, y) \mid Ay = b,\ By + Dx = d,\ x \geq 0,\ y \geq 0 \text{ and integer}\}.$$

Let F be the feasible space for (dualSPBD($\overline{y}$)), i.e., $F = \{\pi \mid \pi^T D \leq c^T,\ \pi \text{ is free}\}$. Note that F is independent of the choice of $\overline{y}$. Assume that F is not empty. The set of extreme points and extreme rays of F are defined by the set E and Q, respectively.

Since we have fixed the value of $\overline{y}$, we can consider the projection of a point $(x, y) \in R$ into the subspace $H^{\overline{y}} = \{(x, \overline{y}) \in R \mid A\overline{y} = b\}$. We denote the projection as $\text{Proj}_{H^{\overline{y}}}(R)$. The projection can be derived by Theorem 3.1.1 (Theorem 4.10 in Nemhauser & Wolsey (1988)).

Theorem 3.1.1. *Let R be the feasible region of* (OriginalBD) *and $H^y = \{(x, y) \in R \mid Ay = b\}$. Then*

$$Proj_{H^y}(R) = \{y \mid r_q^T(d - By) \leq 0 \quad \forall q \in Q,\ y \geq 0 \text{ and integer}\}$$

where $\{r_q\}_{q \in Q}$ are the extreme rays of $Q = \{r \mid r^T D \leq 0,\ r \text{ is free}\}$.

Then, the (OriginalBD) problem can be reformulated as follows:

$$\min_{y \in \text{Proj}_{H^y}(R)} f^T y + z_{LP}(y).$$

If $z_{LP}(y)$ where $y \in \text{Proj}_{H^y}(R)$ has a finite optimal solution, we can represent it in the form of extreme points.

$$z_{LP}(y) = \max_{e \in E} \pi_e^T(d - By)$$

where $\{\pi_e\}_{e \in E}$ are the extreme points of F.

Consequently, we can derive the equivalent formulation for (OriginalBD) below:

$$\begin{aligned}
\min_{y} \quad & f^T y + \{\max_{e \in E} \pi_e^T(d - By)\} \\
\text{s.t.} \quad & Ay = b \\
& r_q^T(d - By) \leq 0 && \forall q \in Q \\
& y \geq 0 \quad \text{and integer.}
\end{aligned}$$

Let $\eta \in \mathbb{R}$ be the continuous variable to represent the value of the optimal extreme point, i.e.,

$$\eta = \max_{e \in E} \pi_e^T(d - By).$$

Finally, we get **Benders' Reformulation** which is an equivalent form of (OriginalBD).

$$\begin{aligned}
\min_{y,\eta} \quad & f^T y + \eta \\
\text{s.t.} \quad & Ay = b && \text{(BR)} \\
& \eta \geq \pi_e^T(d - By) && \forall e \in E \\
& r_q^T(d - By) \leq 0 && \forall q \in Q \\
& y \geq 0 \quad \text{and integer} \\
& \eta \in \mathbb{R}.
\end{aligned}$$

The characterisation of solutions of (BR) is included in Theorem 3.1.2, which is derived from

Theorem 3.1 (Partitioning theorem for mixed-variables programming problems) from Benders (1962).

Theorem 3.1.2. *(Partitioning theorem for mixed-variables programming problems)*

1. *Problem* (OriginalBD) *is not feasible if and only if the* (BR) *is not feasible, i.e., if and only if the feasible region is empty.*
2. *Problem* (OriginalBD) *is feasible without having an optimum solution, if and only if the* (BR) *is feasible without having an optimum solution.*
3. *If* $(\overline{x}, \overline{y})$ *is an optimum solution of problem* (OriginalBD) *and* $f^T\overline{y} + \overline{\eta} = f^T\overline{y} + c^T\overline{x}$*, then* $(\overline{\eta}, \overline{y})$ *is an optimum solution of problem* (BR) *and* $\overline{x}$ *is an optimum solution of the linear programming problem* (SPBD($\overline{y}$)).
4. *If* $(\overline{\eta}, \overline{y})$ *is an optimum solution of* (BR)*, then* (SPBD($\overline{y}$)) *is feasible and the optimum value of the objective function in this problem is equal to* $\overline{\eta}$*. If* $\overline{x}$ *is an optimum solution of* (SPBD($\overline{y}$))*, then* $(\overline{x}, \overline{y})$ *is an optimum solution of problem* (OriginalBD)*, with optimum value* $f^T\overline{y} + \overline{\eta}$ *for the objective function.*

3.2 Classical Benders' Decomposition

Although it is possible to derive Benders' Reformulation, it is very hard to find all extreme points and extreme rays a priori. Moreover, the size of the set of extreme points and extreme rays is usually large. Even if we were able to include all of them in the formulation, they would become a major computational burden. Therefore, we start with the relaxation of (BR), called the master problem, where the corresponding set of extreme points and extreme rays is empty. The master problem is solved for the optimal solution $\overline{y}$. Then, (dualSPBD($\overline{y}$)), which is the dual subproblem corresponding to $\overline{y}$, can be unbounded or have an optimal solution. In the former case, we obtain so-called feasibility cuts; in the latter case, we obtain optimality cuts. The cuts are added to the master problem to either provide the feasibility requirements or improve the quality of the integer solution. The master problem and the subproblem are solved iteratively. The algorithm stops when we find the optimal solution or some other conclusions, for example, infeasibility or unboundedness for (OriginalBD).

According to Nemhauser & Wolsey (1988), a rigorous algorithm for the classical Benders' decomposition in general is shown in Algorithm 1. Initially, the master problem has subsets of extreme points and extreme rays, E^1 and Q^1, respectively, which are usually empty. As the algorithm progresses, the master problem at iteration t is represented by (MPBD^t).

$$z^t = \min_{y,\eta} \{f^T y + \eta \mid (\eta, y) \in S_R^t\} \qquad \text{(MPBD}^t\text{)}$$

where S_R^t is the current feasible region for the master problem corresponding to the set E^t and Q^t, i.e.,

$$S_R^t = \{\eta \in \mathbb{R},\ y \geq 0 \text{ and integer} \mid Ay = b,\ \eta \geq \pi_e^T(d - By) \quad \forall e \in E^t,\ r_q^T(d - By) \leq 0 \quad \forall q \in Q^t\}.$$

Line 4 shows that if the master problem, which is the relaxation of the original problem, is infeasible, then the original problem is also infeasible. For the other cases, shown in Lines 5−9, we can derive a master solution (η^t, y^t). We then solve the corresponding dual problem of (SPBD(y^t)), called the dual subproblem at the iteration t, represented in (dualSPBD(y^t)).

$$z_{LP}(y^t) = z_{LP}^t = \max\{\pi^T(d - By^t) \mid \pi^T D \leq c,\ \pi \text{ free}\}. \qquad \text{(dualSPBD}(y^t)\text{)}$$

If the dual subproblem is infeasible, then the primal subproblem is unbounded. In such a case, (BR) is also unbounded, since there is a feasible master solution that can lead to unboundedness for the original problem. Otherwise, we can derive an extreme ray or extreme point: see Lines 13−17.

Line 18 is the optimality test for the current solution. If it does not pass the test, it means that at least one constraint of (BR) is violated. Lines 21−27 show how to update an optimality cut and a feasibility cut. The algorithm repeats the process until it reaches a stopping criterion,

which can be a restriction of the computational time or the acceptable gap between the upper bound and the lower bound.

Regarding the bounds for (BR), the master problem provides a lower bound, while the subproblem gives an upper bound. Assuming that at iteration t, (η^t, y^t) is the optimal integer solution of (MPBD^t), the objective function of the master problem, $f^T y^t + \eta^t$, is a lower bound since (MPBD^t) is a relaxation of (BR). Whenever we can solve the corresponding (dualSPBD(y^t)) optimally, the optimal subproblem solution x^t is derived easily from the optimal dual solution π^t by the complementary slackness conditions. Now, (x^t, y^t) is a feasible solution for (BR), and, therefore,

$$f^T y^t + {\pi^t}^T (d - B y^t) = f^T y^t + c^T x^t$$

is an upper bound of (BR).

Algorithm 1 The Classical Benders' Decomposition in General (Nemhauser & Wolsey 1988)

Input: $E^1 \subseteq E$, $Q^1 \subseteq Q$
Initial:
$t := 1$
while It does not reach any stopping criterion **do**
  Solve (MPBD^t)
  if (MPBD^t) is infeasible **then**
    Stop. (BR) is infeasible
  else if (MPBD^t) is unbounded **then**
    Find a feasible solution pair (η^t, y^t) with $\eta^t > \omega$ for some large value ω
  else
    Derive the optimal solution (η^t, y^t)
  end if
  Solve (dualSPBD(y^t))
  if (dualSPBD(y^t)) is infeasible **then**
    Stop. (BR) is unbounded
  else if (dualSPBD(y^t)) is unbounded **then**
    Following Farkas' Lemma, we can find an extreme ray r^t
  else if (dualSPBD(y^t)) is finite **then**
    Get the dual solution π^t and the corresponding primal solution x^t
  end if
  if $f^T y^t + \eta^t \geq f^T y^t + c^T x^t$ **then** ▷ Optimality Test
    Stop. (x^t, y^t) is the optimal solution of (BR)
  else if $f^T y^t + \eta^t < f^T y^t + c^T x^t$ **or** (dualSPBD(y^t)) is unbounded **then** ▷ Violation
    if (dualSPBD(y^t)) is finite **then**
      Update set $E^{t+1} = E^t \cup \{\pi^t\}$
      Update $S_R^{t+1} = S_R^t \cap \{(\eta, y) \mid \eta \geq {\pi^t}^T (d - By)\}$ ▷ Update the optimality cut
    else if (dualSPBD(y^t)) is unbounded **then**
      Update set $Q^{t+1} = Q^t \cup \{r^t\}$
      Update $S_R^{t+1} = S_R^t \cap \{(\eta, y) \mid {r^t}^T (d - By) \leq 0\}$ ▷ Update the feasibility cut
    end if
  end if
  Update $t := t + 1$
end while

3.3 Benders' Decomposition for our Model

To apply Benders' decomposition to our model, we separate our mathematical formulation into the master problem and the subproblem in the usual way. We keep all the integer variables (the week centre and week pattern variables) and their associated constraints in the master problem, and the rest in the subproblem. Then, the master problem is a mixed-integer programme, and the subproblem is a linear problem.

We assume that we have an integer solution from the master problem, $(\overline{g}, \overline{x})$, where

- $\overline{g} \equiv (\overline{g}_{bp})_{b\in B, p\in P}$, satisfying Constraints (2.1), (2.5), (2.6), and (2.7) for the unique week pattern per customer and weekly workload balance.
- $\overline{x} \equiv (\overline{x}_b^w)_{b\in B, w\in W}$, satisfying Constraints (2.4) and (2.8) to ensure that there must be exactly one week centre each week.

The Benders' subproblem corresponding to $(\overline{g}, \overline{x})$ is

$$\begin{aligned}
\min \quad & z(u) := \sum_{b\in B}\sum_{i\in B}\sum_{w\in W} c_{bi}u_{bi}^w && && (\mathrm{SP}(\overline{g},\overline{x}))\\
\text{s.t.} \quad & \sum_{i\in B} u_{bi}^w = \sum_{p\in P_b} \psi_p^w \overline{g}_{bp} && b\in B,\ w\in W && (3.1)\\
& u_{bi}^w \le \overline{x}_i^w && b,i\in B,\ w\in W && (3.2)\\
& u_{bi}^w \ge 0 && b,i\in B,\ w\in W. &&
\end{aligned}$$

Let α_b^w and β_{bi}^w be the dual variables for (3.1) and (3.2), respectively. The dual of the Benders' subproblem as follows:

$$\begin{aligned}
\max \quad & z(\alpha,\beta) := \sum_{b\in B}\sum_{w\in W}\sum_{p\in P_b} \alpha_b^w \psi_p^w \overline{g}_{bp} - \sum_{b\in B}\sum_{i\in B}\sum_{w\in W} \beta_{bi}^w \overline{x}_i^w && && (\mathrm{dualSP}(\overline{g},\overline{x}))\\
\text{s.t.} \quad & \alpha_b^w - \beta_{bi}^w \le c_{bi} && b,i\in B,\ w\in W && (3.3)\\
& \alpha_b^w \in \mathbb{R} && b\in B,\ w\in W && (3.4)\\
& \beta_{bi}^w \ge 0 && b,i\in B,\ w\in W. && (3.5)
\end{aligned}$$

After solving the dual subproblem, we generate a cut and add it to the master problem. According to Rahmaniani et al. (2017), feasibility cuts increase the number of iterations without improving the lower bound and they slow down the algorithm. Fortunately, the primal subproblem here is always feasible for every choice of weekly centres and week patterns since the task for the problem is only to calculate the corresponding total centre-based distance. Therefore, we only get optimality cuts for the master problem.

The master problem is presented below.

$$\begin{aligned}
\min \quad & \eta && && (\mathrm{MP})\\
\text{s.t.} \quad & \sum_{p\in P_b} g_{bp} = 1 && b\in B && (3.6)\\
& \sum_{b\in B} x_b^w = 1 && w\in W && (3.7)\\
& \sum_{b\in B, p\in P_b} t_b \psi_p^w g_{bp} \ge (1-\tau^{week})\mu^{week} && w\in W && (3.8)\\
& \sum_{b\in B, p\in P_b} t_b \psi_p^w g_{bp} \le (1+\tau^{week})\mu^{week} && w\in W && (3.9)\\
& \sum_{b\in B}\sum_{w\in W}\sum_{p\in P_b} \alpha_b^w \psi_p^w g_{bp} - \sum_{b\in B}\sum_{i\in B}\sum_{w\in W} \beta_{bi}^w x_i^w \le \eta && (\alpha,\beta)\in \pi^{SP}_{\text{points}} && (3.10)\\
& g_{bp}\in\{0,1\} && b\in B,\ p\in P_b && (3.11)\\
& x_b^w \in \{0,1\} && b\in B,\ w\in W && (3.12)\\
& \eta\in\mathbb{R}. &&
\end{aligned}$$

Constraints (3.6)−(3.9), and (3.11)−(3.12) are for the integer variables from our original model. The optimality cuts are (3.10), where (α,β) denote $\alpha \equiv (\alpha_b^w)_{b\in B, w\in W}$ and $\beta \equiv (\beta_{bi}^w)_{b,i\in B, w\in W}$, and π^{SP}_{points} is the set of extreme points of the feasible region of the dual subproblem.

We derive bounds for the problem in the way that we described in the previous section. At iteration t, let (η^t, g^t, x^t) be the optimal solution of the master problem. η^t, which is the objective value of the master problem, provides the lower bound. If the corresponding

dual subproblem is solved optimally, we can derive the optimal solution of the subproblem $u^t \equiv (u_{bi}^w)_{b,i\in B,w\in W}$. Then, (g^t, x^t, u^t) is a feasible solution for the original problem and gives an upper bound. In particular, the objective value of the subproblem is already an upper bound since its objective function is the total centre-based distance corresponding to the predetermined week centres and week patterns.

Finally, there are several potential criteria to stop the algorithm.

1. The gap between the lower bound and upper bound is within a predefined tolerance. In this work, we consider the relative percentage gap. Let UB and LB be the upper bound and the lower bound of the problem, respectively. The relative percentage gap P_G is given as
$$P_G = \begin{cases} \frac{(UB-LB)\times 100}{UB}, & UB > 0, \\ 100, & \text{otherwise.} \end{cases} \tag{3.13}$$
2. The limitation in the total time to run the algorithm.
3. The original model is infeasible or unbounded.

The complete algorithm of Benders' decomposition for our model is presented in Algorithm 2, where $maxTime$ and ϵ are the parameters for the stopping criteria: the maximum time and the threshold of the acceptable relative percentage gap, respectively.

In iteration t, the master problem is solved for an integer solution. Unless the master problem is infeasible, we derive an integer solution and update LB, the lower bound for the problem. Afterwards, the dual subproblem corresponding to the integer solution is solved. Since the task for the subproblem is just to calculate the total centre-based distance for the predefined week patterns and week centres, it is always solved optimally. It generates an optimality cut and updates the upper bound UB: see Lines 12−15. Regarding the stopping criteria for the algorithm, Lines 16 and 17 update the relative percentage gap P_G and the current time $Time$, respectively.

Algorithm 2 The Classical Benders' Decomposition Algorithm

Parameter: $maxTime$, ϵ
Initial:
LB:= 0, UB:=1e10, t:=0, $Time$:= 0, π^{SP}_{points} := $\emptyset$
1: **while** $Time < maxTime$ **and** $P_G > \epsilon$ **do**
2: $t := t + 1$ ▷ Update the number of iterations
3: Solve (MP)
4: **if** (MP) is infeasible **then**
5: (OriginalSchedule) is infeasible. **Stop the algorithm**
6: **else if** (MP) is unbounded **then**
7: Find any reasonable feasible solutions (g^t, x^t)
8: **else**
9: Get the optimal solution of (MP), (g^t, x^t)
10: Update LB:= $\max\{LB, \eta^t\}$
11: **end if**
12: Solve (dualSP(g^t, x^t))
13: Derive (α, β) and update the set of π^{SP}_{points}
14: Generate the corresponding optimality cut (3.10)
15: Update UB:= $\min\{UB, z(\alpha, \beta)\}$
16: Update P_G by (3.13)
17: Record current time in $Time$
18: **end while**

However, there are several well-known drawbacks of the classical Benders' decomposition. First, the master problem always provides a poor quality of integer solutions in the beginning since there is not much information from the subproblem. It results in a slow increase in the lower bound. Secondly, there is no guarantee in the strength of the cuts from the subproblem, so it is possible to provide poor cuts for the master problem. Moreover, we solve the master problem, an integer problem, with more constraints from scratch in each iteration. As a result, it becomes a major computational burden as the algorithm progresses (Rahmaniani et al. 2017).

Therefore, in the next section, we propose acceleration techniques to improve the efficiency of Benders' decomposition.

3.4 Accelerating Benders' Decomposition

In this section, we show the various developments on the classical Benders' decomposition to improve its efficiency for solving our model. To strengthen the LP relaxation of the master problem, thereby enhancing the efficiency of generating the integer solution, we apply two-phase Benders' decomposition. Moreover, we use a high-quality integer solution from a heuristic to generate optimality cuts beforehand. This is to improve the upper bound of the objective value at the beginning. Regarding optimality cuts in the algorithm, we can derive them by hand because of the particular structure of the subproblem. However, the cuts might not be the strongest, since the subproblem is highly degenerate. Namely, the dual subproblem can have several optimal solutions that produce cuts of different strength. A *nondominated optimality cut* is used in such a case as it is guaranteed to be the strongest cut. We also exploit the decomposable structure of the subproblem to generate multiple cuts per iteration. Finally, we propose a modern implementation called the *branch-and-Benders' cut*. Instead of solving the master problem from scratch in every iteration, we consider a single branch-and-bound tree of the master problem and generate Benders' cuts from a node in the tree.

3.4.1 Two-Phase Benders' Decomposition

As we mentioned above, the master problem always struggles to provide a good quality of integer solutions in the first iterations since it lacks the information from the subproblem. McDaniel & Devine (1977) addressed this problem and proposed an idea to improve the quality of solutions of the master problem, called the *two-phase Benders' decomposition*. They separate a problem into two phases: the *relaxed Benders' decomposition* and the *Benders' decomposition*. In the first phase, they relax the integrality conditions in the master problem and generate Benders' cuts from fractional solutions. This leads to a tighter linear programming relaxation for the master problem. Then, they apply the classical Benders' decomposition in the second phase by adding the integrality conditions back into the master problem. This algorithm is effective and easy to implement, so it is popular in the literature (for example, Papadakos (2009), de Sá et al. (2013), Botton et al. (2013), Kergosien et al. (2017), de Sá et al. (2018)).

We have two criteria for ending the first phase: reaching the predetermined maximum number of iterations to add cuts and the relative percentage gap of improvement. The first criterion is to make sure that we will not add too many cuts before starting the next phase. The latter one considers the relative percentage gap of the objective value of the master problem between two consecutive iterations. If the relative percentage gap is under the predetermined threshold, it suggests that the recent cuts are not very helpful. A way to calculate this measurement is represented below.

Let η^t be the objective value of the master problem at iteration t. P_{Imp} is the relative percentage gap of the improvement for $t = 2, 3, \ldots$ as follows:

$$P_{Imp} = \begin{cases} \frac{(\eta^t - \eta^{t-1}) \cdot 100}{\eta^{t-1}}, & \eta^{t-1} > 0,\ t = 2, 3, \ldots, \\ 100, & \text{otherwise.} \end{cases} \tag{3.14}$$

When any of these stopping criteria is reached, the algorithm moves to the second phase and follows the steps in Algorithm 2.

Algorithm 3 shows the additional first phase. We introduce new parameters for the specific stopping criteria of the first phase: N is the maximum number of iterations to add optimality cuts from a fractional solution, and ϵ^{1st} is the acceptable improvement tolerance of the relative percentage gap on the objective value of the master problem. The modified algorithm starts by creating the LP relaxation of the master problem, called (MPRelax). Then, most of the steps in the first phase are almost the same as the classic version. Note that all solutions of the master problem in the first phase are fractional and, thus, not feasible for the original problem. Therefore, we cannot update the upper bound of the algorithm in the first phase.

Line 4 updates the relative percentage gap of the improvement for the stopping criterion. When we finish the first phase, we reintroduce the integrality conditions in the master problem. Also, the total number of iterations, t, is reinitialised. Line 8 updates the current time in $Time$ to include the total computational time of the first phase into the second phase. Usually, the first phase finishes quickly since it deals only with a linear programme. Then, the algorithm continues with the procedure of the classical version.

Algorithm 3 Two-Phase Benders' Decomposition

Parameter:
For the first phase : N, ϵ^{1st}
For the second phase : $maxTime$, ϵ
Initial:
LB:= 0, UB:=1e10, t:=0, $Time := 0$, $\pi^{SP}_{points} := \emptyset$
1: Relax the integrality conditions, (3.11) and (3.12), in (MP). We call it (MPRelax)
2: **while** $P_{Imp} > \epsilon^{1st}$ **and** $t < N$ **do**
3: Follow Lines 2−14 of Algorithm 2
▷ Solve (MPRelax) and the dual subproblem, and derive the optimality cut
4: Update P_{Imp} by using (3.14)
5: **end while**
6: Reintroduce the integrality conditions, (3.11) and (3.12), in (MP)
7: $t := 0$ ▷ Reinitialise the number of iterations for the next phase
8: Update the current time in $Time$ ▷ Include the solving time for the first phase
9: Run Algorithm 2

3.4.2 Initial Integer Solution

The master problem determines the week pattern of each customer and the week centres by means of the week pattern and the week centre variables, respectively. To generate a high-quality initial solution for the master problem, we adopt Cooper's location-allocation heuristic from Cooper (1964).

The heuristic is usually applied in a facility location problem to decide on warehouse locations and allocate customers to those warehouses (for example, Hess & Samuels (1971)). It splits the problem into two separate problems: the location problem and the allocation problem. The location problem determines optimal warehouse locations, given that the allocations of customers to warehouses are fixed. The allocation problem optimally allocates the customers to the fixed warehouse locations. The two problems are solved repeatedly until both of them reach the same objective value. Transferred to the context of districting problems, customers correspond to basic units and warehouses to district centres. The heuristic is usually quick to solve and provides a high-quality solution. Bender et al. (2016) demonstrated its success in solving the multi-period sales districting problem for large data instances.

Here we design the location problem for the week centre variables and the allocation variables, where the values of the week pattern variables are fixed. Regarding the allocation problem, we determine the week patterns of customers for fixed values of the centre variables.

For the location problem with fixed week pattern values, we know all customers who require a service each week from their fixed week patterns. Hence, we determine the week centres that minimise the total centre-based distance for all week clusters. Let $\overline{g}$ be a matrix of fixed values of the feasible week pattern variables that satisfy Constraints (3.6), (3.8), (3.9), and (3.11).

The location problem corresponding to $\overline{g}$ is represented in the following problem (Location($\overline{g}$)).

$$\begin{aligned}
\min \quad & z_L := \sum_{b\in B}\sum_{i\in B}\sum_{w\in W} c_{bi}u_{bi}^{w} && && \text{(Location}(\overline{g})\text{)}\\
\text{s.t.} \quad & \sum_{i\in B} u_{bi}^{w} = \sum_{p\in P_b} \psi_p^w \overline{g}_{bp} && b\in B,\ w\in W\\
& \sum_{b\in B} x_b^w = 1 && w\in W\\
& u_{bi}^{w} \le x_i^w && b,i\in B,\ w\in W\\
& u_{bi}^{w} \ge 0 && b,i\in B,\ w\in W\\
& x_b^w \in \{0,1\} && b\in B,\ w\in W.
\end{aligned}$$

This problem is very easy to solve by hand since the only remaining task is to find a customer who should be the week centre each week, i.e., which customer provides the minimum total centre-based distance in each week cluster. Corresponding to the information of $\overline{g}$, let $B_w = \{b \in B \mid \sum_{p\in P_b} \psi_p^w \overline{g}_{bp} > 0\}$ denote the set of customers in week $w \in W$. The week centre of week w, b^w, can be derived by solving the following:

$$b^w = \arg\min_{i\in B} \sum_{b\in B_w} c_{bi}. \tag{3.15}$$

Then, we can derive the values of the centre variables immediately from the following equations (3.16).

$$x_b^w = \begin{cases} 1, & b = b^w,\\ 0, & \text{otherwise;} \end{cases} \tag{3.16}$$

These week centres and the set of customers in each week imply the corresponding allocation variables values in (3.17) as shown below.

$$u_{bi}^{w} = \begin{cases} 1, & b\in B_w \text{ and } i = b^w,\\ 0, & \text{otherwise.} \end{cases} \tag{3.17}$$

Afterwards, we can derive the location objective value z_L.

Regarding the allocation problem, assume that we have centre variable values $\bar{x}$ that satisfy (3.7) and (3.12). The objective function for the problem is a little bit different from the original one. The total centre-based measurement in this problem is based on the week pattern variables and the distance between each customer and the predetermined week centres. The task is to select a week pattern for each customer that minimises the total centre-based distance for the fixed week centres.

Let $c_b^w(\overline{x})$ be the distance between customer $b \in B$ and the week centre in week w corresponding to $\overline{x}$, i.e.,

$$c_b^w(\overline{x}) = \begin{cases} c_{bi}, & \text{where } \overline{x}_i^w = 1,\ i\in B,\\ 0, & \text{otherwise.} \end{cases}$$

The allocation problem for a given $\overline{x}$ is shown in (Allocation($\overline{x}$)) as follows.

$$\min \quad z_{AL} := \sum_{b\in B}\sum_{w\in W}\sum_{p\in P_b} c_b^w(\overline{x})\psi_p^w g_{bp} \qquad \text{(Allocation}(\overline{x})\text{)}$$

$$\text{s.t.} \quad \sum_{p\in P_b} g_{bp} = 1 \qquad b \in B$$

$$\sum_{b\in B, p\in P_b} t_b\psi_p^w g_{bp} \geq (1-\tau^{week})\mu^{week} \qquad w \in W$$

$$\sum_{b\in B, p\in P_b} t_b\psi_p^w g_{bp} \leq (1+\tau^{week})\mu^{week} \qquad w \in W$$

$$g_{bp} \in \{0,1\} \qquad b \in B,\ p \in P_b.$$

To start the location-allocation heuristic, we create an initial matrix of week centre values by picking a customer randomly to be a week centre each week. We restrict ourselves to selecting customers for the random week centres who have the smallest week rhythm in the planning horizon. The reason is that they are most likely to appear in many week clusters.

Let $\overline{r} = \min_{b\in B}\ r_b$ be the smallest week rhythm of the customers and let set $B^{\overline{r}}$ represent the set of customers whose week rhythm is $\overline{r}$. In particular, $B^{\overline{r}} = \{b \in B \mid r_b = \overline{r}\}$. In week $w \in W$, we randomly select $b^w \in B^{\overline{r}}$ as the week centre. Then, we use (3.16) to derive the corresponding matrix for the week centre variable values $\overline{x}$. The heuristic starts by solving the allocation problem with $\overline{x}$, and it continues solving the location and the allocation problem alternatingly until both objective values reach the same value. This yields an initial feasible integer master solution (g^0, x^0). Moreover, we generate an optimality cut from this solution and add it at the beginning of Benders' decomposition algorithm. Since the heuristic usually generates a high-quality solution, this optimality cut tends to tighten the feasible region of the master problem effectively.

3.4.3 Manual Derivation of an Optimality Cut from an Integer Master Solution

Since week centres and week patterns for customers have been determined already in the master problem, we can directly derive the allocation variables' values and calculate the compactness of the schedules.

Assuming that $(\overline{g}, \overline{x})$ is an integer solution of the master problem, we derive set B_w of customers requiring service in week w, i.e., $B_w = \{\ b \in B \mid \sum_{p\in P_b} \psi_p^w \overline{g}_{bp} > 0\ \}$. Concerning the week centres, let $i(w)$ be the customer who is the week centre in week w, i.e., $\overline{x}_{i(w)}^w = 1$. Then, the values of the allocation variables can be derived immediately as follows:

$$u_{b,i(w)}^w = 1 \qquad b \in B_w$$
$$u_{bi}^w = 0 \qquad b \in B_w,\ i \in B \setminus \{i(w)\}$$
$$u_{bi}^w = 0 \qquad b \notin B_w,\ i \in B.$$

Let us consider the complementary slackness conditions below.

$$(\sum_{i\in B} u_{bi}^w - \sum_{p\in P_b} \psi_p^w \overline{g}_{bp}) \cdot \alpha_b^w = 0 \qquad b \in B,\ w \in W \qquad (3.18)$$

$$(\overline{x}_i^w - u_{bi}^w) \cdot \beta_{bi}^w = 0 \qquad b, i \in B,\ w \in W \qquad (3.19)$$

$$(\alpha_b^w - \beta_{bi}^w - c_{bi}) \cdot u_{bi}^w = 0 \qquad b, i \in B,\ w \in W. \qquad (3.20)$$

Then, we can derive an optimal solution for the dual subproblem by inspection from the conditions (3.18)−(3.20) as follows.

For customer $b \in B_w$ in week $w \in W$, the corresponding $u_{b,i(w)}^w = 1$. From (3.20), we get

$$\alpha_b^w - \beta_{b,i(w)}^w = c_{b,i(w)} \qquad (3.21)$$

The objective function of (dualSP($\overline{g}, \overline{x}$)) shows that the coefficients of $\{\alpha_b^w\}_{b\in B, w\in W}$ are positive, while those of $\{\beta_{bi}^w\}_{b,i\in B, w\in W}$ are negative which they are by definition of Constraints (3.5). To optimise the objective function, we choose the maximum possible positive values of α_b^w and the minimum possible values of β_{bi}^w. Therefore, according to (3.21), we select

$$\begin{aligned} \alpha_b^w &= c_{b,i(w)} \\ \beta_{b,i(w)}^w &= 0. \end{aligned} \tag{3.22}$$

For the rest of β_{bi}^w where $i \in B \setminus \{i(w)\}$, their feasible values are according to (3.3), (3.22), and (3.5), i.e.,

$$\beta_{bi}^w \geq c_{b,i(w)} - c_{bi} \text{ and } \beta_{bi}^w \geq 0.$$

Therefore, their possible minimum values are

$$\beta_{bi}^w = \max\{0,\ c_{b,i(w)} - c_{bi}\} = (c_{b,i(w)} - c_{bi})^+.$$

For customer $b \notin B_w$ in week $w \in W$ and customer $i \in B$, feasible dual values that obviously satisfy the complementary slackness conditions (3.18)−(3.20) are

$$\alpha_b^w = \beta_{bi}^w = 0.$$

In conclusion, a solution of the dual subproblem that we derive manually is shown below.

$$\alpha_b^w = c_{b,i(w)} \qquad b \in B_w \tag{3.23}$$

$$\beta_{bi}^w = \max\{0,\ c_{b,i(w)} - c_{bi}\} = (c_{b,i(w)} - c_{bi})^+ \qquad b \in B_w,\ i \in B \tag{3.24}$$

$$\alpha_b^w = \beta_{bi}^w = 0 \qquad b \notin B_w,\ i \in B. \tag{3.25}$$

Then, we can create an optimality cut for the master problem directly, i.e.,

$$\sum_{w\in W} \sum_{b\in B_w} \left(c_{b,i(w)} \sum_{p\in P_b} \psi_p^w g_{bp} - \sum_{i\in B} (c_{b,i(w)} - c_{bi})^+ x_i^w \right) \leq \eta. \tag{3.26}$$

However, we can derive more than one dual solution for the dual subproblem. In particular, the objective function of the dual subproblem does not take into account the customers who do not require service each week and those who are not the week centres. Therefore, we can change the dual values associated with those customers and still get the same objective value.

Namely, for customer $b \notin B_w$, $w \in W$, the dual variables α_b^w do not appear in the dual subproblem objective function because their corresponding coefficient values are zero, i.e., $\sum_{p\in P_b} \psi_p^w \overline{g}_{bp} = 0$. Similarly to β_{bi}^w, $i \in B \setminus \{i(w)\}$, their coefficients correspond to $\overline{x}_i^w = 0$. Therefore, the values of α_b^w and β_{bi}^w in (3.25) can be changed to any values that satisfy Constraints (3.3)−(3.5), without changing the dual subproblem objective value. For example, we can increase the values of these α_b^w and β_{bi}^w to be similar to those for the customers who require service in (3.23) and (3.24), respectively.

$$\alpha_b^w = c_{b,i(w)} \qquad b \notin B_w$$

$$\beta_{bi}^w = \max\{0,\ c_{b,i(w)} - c_{bi}\} = (c_{b,i(w)} - c_{bi})^+ \qquad b \notin B_w,\ i \in B.$$

These are feasible for the dual subproblem and do not increase the objective value of the dual subproblem.

The dual subproblem is highly degenerate as we showed in the above example. Therefore, (3.26) might not be the strongest possible cut, and this can lead to a potentially weak improvement of the bounds. In the next section, we use the concept of a nondominated optimality cut to derive the strongest cut in each iteration.

3.4.4 Nondominated Optimality Cut

According to Section 3.2, for a given integer solution $\overline{y}$, we solve the corresponding dual subproblem (dualSPBD($\overline{y}$)), i.e.,

$$z_{LP}(\overline{y}) := \max\{\pi^T(d - B\overline{y}) \mid \pi^T D \leq c^T,\ \pi \text{ is free}\}$$

to generate a Benders' cut.

If we can solve the dual subproblem to optimality, we will derive a dual solution π to generate an optimality cut as follows:

$$\eta \geq \pi^T(d - By). \tag{3.27}$$

In case that the dual subproblem is degenerate, it is possible to derive more than one optimal solution. As a result, we might generate an optimality cut with different strength. To guarantee the efficiency of the algorithm, we aim for generating the strongest optimality cut in every iteration. A few important approaches are presented in this subsection.

Pareto-Optimal Cut by Magnanti & Wong (1981)

Magnanti & Wong (1981) defined the strongest cut or a Pareto-optimal cut as shown below.

Definition 1. *A cut $\eta \geq \pi^{*T}(d - By)$ **dominates** or is **stronger** than $\eta \geq \pi^{0T}(d - By)$, if $\pi^{*T}(d - By) \geq \pi^{0T}(d - By)$ for all $y \in Y$ with a strict inequality for at least one point $y \in Y$, where Y is the set of all feasible solutions of the master problem*

Definition 2. *A cut is **Pareto-optimal** if it is not dominated by any other cut.*

To derive Pareto-optimal cuts for a given master solution, Magnanti & Wong (1981) suggested solving the subproblem twice. The first subproblem is the standard subproblem whose objective value is used to derive a set of alternative optimal solutions. The second subproblem determines a Pareto-optimal cut among these solutions. In particular, the second one is solved using a so-called *core point*. A core point is any point in the relative interior of the convex hull of the set of feasible points of the master problem.

Let $\overline{y}$ and $z_{LP}(\overline{y})$ be a given master solution and the objective value of the corresponding standard dual subproblem, respectively. Let y^0 be a core point. The formulation of the second dual subproblem is below.

$$\begin{aligned} \max \quad & \pi^T(d - By^0) && \text{(MWGeneral}(y^0)\text{)} \\ \text{s.t.} \quad & \pi^T(d - B\overline{y}) = z_{LP}(\overline{y}) && (3.28) \\ & \pi^T D \leq c^T \\ & \pi \text{ is free.} \end{aligned}$$

There are few differences to the standard subproblem: the core point substitutes the integer solution in the objective function, and (3.28) is an additional constraint to guarantee that a dual solution here has the same objective value as in the standard subproblem. According to Theorem 1 from Magnanti & Wong (1981), the optimal dual solution from the above problem generates a Pareto-optimal cut.

To adapt the cut generation approach to our model, let (g^0,x^0) be a core point for the week pattern and the week centre variables, and ($\overline{g}$,$\overline{x}$) be an integer solution from the Benders' master problem (MP). After solving (dualSP($\overline{g}$,$\overline{x}$)) for the corresponding dual solution ($\overline{\alpha}$,$\overline{\beta}$), we then solve the following problem (MW(g^0,x^0)) which is a modified subproblem for the core

point.

$$\max \quad z(\alpha,\beta) := \sum_{b\in B}\sum_{w\in W}\sum_{p\in P_b} \alpha_b^w \psi_p^w g_{bp}^0 - \sum_{b\in B}\sum_{i\in B}\sum_{w\in W} \beta_{bi}^w x_i^{0w} \qquad (\mathrm{MW}(g^0,x^0))$$

$$\text{s.t.} \quad \sum_{b\in B}\sum_{w\in W}\sum_{p\in P_b} \alpha_b^w \psi_p^w \overline{g}_{bp} - \sum_{b\in B}\sum_{i\in B}\sum_{w\in W} \beta_{bi}^w \overline{x}_i^w = z(\overline{\alpha},\overline{\beta}) \qquad (3.29)$$

$$\alpha_b^w - \beta_{bi}^w \le c_{bi} \qquad b,i \in B, w \in W$$

$$\alpha_b^w \in \mathbb{R} \qquad b \in B,\ w \in W$$

$$\beta_{bi}^w \ge 0 \qquad b,i \in B,\ w \in W.$$

The modified algorithm is represented in Algorithm 4. It shows the additional steps after solving the standard subproblem in Lines 4−7. A core point (g^0, x^0) is generated. Then, the core point and the dual solution of the standard subproblem (α, β) are for creating $(\mathrm{MW}(g^0, x^0))$. The solution (α', β') from $(\mathrm{MW}(g^0, x^0))$ is used to generate the Pareto-optimal cut for the master problem.

Algorithm 4 Magnanti-Wong Algorithm

Parameter: $maxTime$, ϵ
Initial:
LB:= 0, UB:=1e10, t:=0, $Time$:= 0 π_{points}^{SP} := $\emptyset$
1: **while** $Time < maxTime$ **and** $P_G > \epsilon$ **do**
2: Follow Lines 2−12 in Algorithm 2 ▷ Solve the master and the dual subproblem
3: Get the corresponding dual optimal solution (α,β)
4: Find a core point (g^0,x^0)
5: Use (g^0,x^0) and (α,β) to create $(\mathrm{MW}(g^0, x^0))$
6: solve $(\mathrm{MW}(g^0, x^0))$ for (α',β') and update the set of π_{points}^{SP}
7: Generate the Pareto-optimality cut (3.10) from (α',β')
8: Update UB:= $\min\{UB, z(\alpha,\beta)\}$
9: Update the relative percentage gap P_G and the recent time in $Time$ like in Algorithm 2
10: **end while**

Modified Pareto-Optimal Cut by Papadakos (2008)

Papadakos (2008) addressed a main drawback of the idea from Magnanti & Wong (1981): Constraint (3.28) in $(\mathrm{MWGeneral}(y^0))$ is a major cause for numerical instability, especially when the subproblem is difficult to solve optimally. The author proved that with a new core point in every iteration, the problem $(\mathrm{MWGeneral}(y^0))$ without Constraint (3.28) can still generate Pareto-optimal cuts. Namely, Constraint (3.28) can be eliminated from problem $(\mathrm{MWGeneral}(y^0))$ to avoid the numerical issue, thereby resulting in an independent problem $(\mathrm{IndMWGeneral}(y^0))$, as shown below.

$$\max \quad \pi^T(d - By^0) \qquad (\mathrm{IndMWGeneral}(y^0))$$

$$\text{s.t.} \quad \pi^T D \le c^T$$

$$\pi \text{ is free.}$$

Moreover, $(\mathrm{IndMWGeneral}(y^0))$ no longer requires the master problem and the dual subproblem to be solved beforehand, since it is independent of the dual subproblem objective value. Therefore, given a core point, we can create a Pareto-optimal cut before solving the master problem to improve the quality of a solution from the master problem. In particular, at the beginning of the algorithm, we can add a Pareto-optimal cut from a core point to the master problem for a head start.

In our Benders' decomposition, the independent second subproblem is as follows.

$$\max \quad z(\alpha,\beta) := \sum_{b\in B}\sum_{w\in W}\sum_{p\in P_b} \alpha_b^w \psi_p^w g_{bp}^0 - \sum_{b\in B}\sum_{i\in B}\sum_{w\in W} \beta_{bi}^w x_i^{0w} \qquad \text{(IndMW}(g^0,x^0)\text{)}$$

$$\text{s.t.} \quad \alpha_b^w - \beta_{bi}^w \le c_{bi} \qquad b,i \in B,\ w \in W$$

$$\alpha_b^w \in \mathbb{R} \qquad b \in B,\ w \in W$$

$$\beta_{bi}^w \ge 0 \qquad b,i \in B, w \in W.$$

Furthermore, Papadakos (2008) recommended a way to derive a new core point in each iteration. It is simply the convex combination between the current solution of the master problem and the previously used core point. Therefore, it is enough to have only one valid initial core point at the start.

At iteration t where $t \ge 1$, let (g^t, x^t) and $(g^{0,t-1}, x^{0,t-1})$ be a current integer solution from the master problem and a previously used core point, respectively. The new core point for the week centre and week pattern variables is given below.

$$g^{0,t} := \frac{1}{2}g^t + \frac{1}{2}g^{0,t-1} \tag{3.30}$$

$$x^{0,t} := \frac{1}{2}x^t + \frac{1}{2}x^{0,t-1}. \tag{3.31}$$

To get a valid initial core point ($g^{0,0}$, $x^{0,0}$), we create a number of feasible integer solutions and calculate the average value for each integer variable. As we generate the initial core point from a set of feasible solutions, $(g^{0,0}, x^{0,0})$ is a point in the relative interior of the convex hull. To create a feasible solution quickly, we first randomly select a customer $b^w \in B$ as the week centre for week $w \in W$. Then, we derive the corresponding week centre value matrix $\overline{x}$ by Equations (3.16) and solve (Allocation($\overline{x}$)) for the optimal week pattern values. Each feasible solution is different as we have different week centre values each time, i.e., no solution has identical week centres for the whole planning horizon.

The way to generate the initial core point is presented in Algorithm 5, where N^g is the predefined total number of feasible integer solutions. Note that in Line 3, in case a set of week centres is identical to that of any previous solutions, we discard this set and create a new one. Finally, the initial core point is derived from the average of the integer feasible solutions: see Lines 6 and 7.

Algorithm 5 Method to Generate an Initial Core Point

Parameter: N^g
Initial:
t:=0
1: **while** $t < N^g$ **do**
2: $t := t + 1$
3: Generate x^t by randomly selecting the week centres which cannot be entirely identical to those of the previous feasible solutions
4: Solve (Allocation(x^t)) for g^t
5: **end while**
6: $x^{0,0} := \frac{\sum_t x^t}{N^g}$
7: $g^{0,0} := \frac{\sum_t g^t}{N^g}$
Output: $(x^{0,0}, g^{0,0})$

For problems where we cannot find any core points easily (for example, Mercier et al. (2005) and Papadakos (2009)), Papadakos (2008) also suggested an alternative point, called *Magnanti-Wong point*, for a substitution. It is capable of generating a Pareto-optimal cut, although it is not a real core point or not even a feasible point for the master problem. However, this point can be derived for only a specific problem structure. The interested reader is referred to Papadakos (2008) for more information.

The independent Magnanti-Wong algorithm is presented in Algorithm 6. The difference

between this version and Magnanti & Wong (1981) is that a Pareto-optimal cut can be generated from a core point before solving the master problem. This allows the algorithm to generate a Pareto-optimal cut from an initial core point for a head start, shown in Lines 1−3. Similarly, Lines 9−11 create a new core point and generate the corresponding Pareto-optimal cut, before solving the master problem in the next iteration.

Algorithm 6 Independent Magnanti-Wong

Parameter: $maxTime$, ϵ
Initial:
LB:= 0, UB:=1e10, t:=0, $Time := 0$, $\pi^{SP}_{points} := \emptyset$
1: Generate an initial core point $(g^{0,0}, x^{0,0})$ and solve (IndMW(g^0, x^0)) for (α', β')
2: Use (α', β') to update set π^{SP}_{points}
3: Generate a Pareto-optimal cut (3.10) from (α',β')
4: Update the current time in $Time$
5: **while** $Time < maxTime$ **and** $P_G > \epsilon$ **do**
6: $t := t + 1$
7: Follow Lines 3−12 in Algorithm 2 ▷ Solve (MP) and (dualSP(g^t, x^t)) like the classical version
8: From the dual solution (α, β) from (dualSP(g^t, x^t)), we update UB:= min$\{UB, z(\alpha, \beta)\}$
9: Update an approximate core point $(g^{0,t}, x^{0,t})$ by equations (3.30) and (3.31), and solve (IndMW(g^0, x^0)) for (α', β')
10: Use (α',β') to update the set of π^{SP}_{points}
11: Generate a Pareto-optimal cut (3.10) from (α',β')
12: Update the relative percentage gap P_G and the current time in $Time$ like in Algorithm 2
13: **end while**

Maximal Nondominated Cut by Sherali & Lunday (2013)

Although Papadakos (2008) suggested the ways to deal with core points, those approaches are still not always applicable for complicated problems. First, it is not always easy for these problems to derive even one valid core point. In such a case, Papadakos (2008) introduced a Magnanti-Wong point, which is not necessarily a real core point, as an alternative point to generate Pareto-optimal cuts. Nevertheless, this point is limited to only a specific problem structure. Therefore, finding core points is still the main difficulty to generate the Pareto-optimal cuts.

Another possible implementation issue in Magnanti & Wong (1981) and Papadakos (2008) comes from having to solve two subproblems in every iteration. Although Pareto-optimal cuts tend to reduce the total number of iterations, this might not compensate for the effort to solve the increasing number of linear programmes in each iteration (Mercier & Soumis 2007).

Sherali & Lunday (2013) addressed these main issues by proposing a *maximal nondominated cut*. This cut does not require any core points. Also, the algorithm solves only one linear programme in each iteration.

Let $\Pi = \{\pi \mid \pi^T D \leq c^T, \pi \text{ is free}\}$ be the feasible region of the dual subproblem. For a given master solution $\overline{y}$, the set of alternative optimal solutions corresponding to the master solution is represented by $\Pi^{alt}(\overline{y})$, i.e.,

$$\Pi^{alt}(\overline{y}) = \{\pi \in \Pi \mid \pi^T(d - B\overline{y}) = z_{LP}(\overline{y})\}.$$

Sherali & Lunday (2013) rewrote an optimality cut (3.27) in the following form.

$$\eta \geq \pi^T d + \sum_{j=1}^{n} (-\pi^T b_j) y_j$$

where b_j is the jth column of matrix B which has n columns in total, and y_j is a master variable at index j.

Then, a maximal nondominated optimality cut is described in the following definition.

Definition 3. *Given a master solution $\overline{y}$, a cut $\eta \geq \pi^T(d - By)$ is nondominated, or more distinctly, is* ***maximal*** *if there does not exist any $\pi^* \in \Pi^{alt}(\overline{y})$ for which $\pi^{*T}d \geq \pi^T d$ and $(-\pi^{*T}b_j) \geq (-\pi^{*T}b_j)$, for $j = 1, 2, \ldots, n$, with at least one of these $(n + 1)$ inequalities being strict.*

This nondominated cut has a less strict definition than a Pareto-optimal cut. It is clear that a Pareto-optimal cut by Magnanti & Wong (1981) and Papadakos (2008) is maximal given that a core point consists of positive values. However, it is not always true for the other way round.

Sherali & Lunday (2013) formulated a way to generate a maximal nondominated cut by means of the following multiple objective linear programmes.

$$\max\{\pi^T d, -\pi^T b_1, -\pi^T b_2, \ldots, -\pi^T b_n \mid \pi \in \Pi^{alt}(\overline{y})\}.$$

We can solve the above multiple objective linear programmes by solving a positive weighted sum of the multiple objective functions, as shown below.

$$\max\{\pi^T d + \sum_{j=1}^{n}(-\pi^T b_j)\hat{y}_j \mid \pi \in \Pi^{alt}(\overline{y})\} \tag{3.32}$$

where $\hat{y} \in \mathbb{R}^n$ is a positive weight vector and $\hat{y}_j$ is its value at index j, .

Now, in order to generate a maximal nondominated cut, we do not need a core point anymore. Therefore, we can select $\hat{y}$ to be any positive weight vector, for example, $\hat{y}_j = 1$ for $j = 1, 2, \ldots, n$. Note that if $\hat{y}$ is a core point, (3.32) is the same as (MWGeneral(y^0)), where $y^0 = \hat{y}$. In such a case, a generated optimality cut will be both maximal and Pareto-optimal.

Moreover, Sherali & Lunday (2013) asserted that solving two subproblems in every iteration, like in Magnanti & Wong (1981) and Papadakos (2008), might decrease the efficiency of the algorithm. This is particularly for problems that require significant computational effort to solve the subproblem. Therefore, the authors combined those two subproblems into a preemptive priority multiple objective programme framework. Namely, the priority here is to solve the standard dual subproblem. Next, it focuses on selecting alternative solutions to derive the strongest cut. The formulation of the preemptive priority multiple objective programmes is presented in (3.33).

$$\max\{\pi^T(d - B\overline{y}) \gg \pi^T(d - B\hat{y}) \mid \pi \in \Pi\} \tag{3.33}$$

According to Sherali & Soyster (1983), we can equivalently represent (3.33) as a weighted-sum problem where $\mu > 0$ is a positive weight that is sufficiently small, as shown below.

$$\max\{\pi^T(d - B\overline{y}) + \mu\pi^T(d - B\hat{y}) \mid \pi \in \Pi\}. \tag{3.34}$$

This can be seen as a small perturbation on the right-hand side of the subproblem where μ is a magnitude of the perturbation.

However, it is not usually practical to derive the weight μ to guarantee the equivalence of (3.33) and (3.34) (Sherali & Lunday 2013, Oliveira et al. 2014). Instead, Sherali & Lunday (2013) provided an alternative value of μ that generates an ϵ_0-optimal nondominated maximal Benders' cut in (3.35).

$$\mu = \frac{\epsilon_0}{M\theta} \tag{3.35}$$

where ϵ_0 is a predefined acceptance level on the absolute optimality gap; M is a sufficiently large penalty for a violation of any constraints of the subproblem. $\theta = \epsilon_0 + \max\{0, \max_i\{\hat{d}_i\}\} - \min\{0, \min_i\{\hat{d}_i\}\}$ where $\hat{d} \equiv d - B\hat{y}$.

Oliveira et al. (2014) further developed how to derive a proper value of μ in practice. At the beginning of the algorithm, the dual solutions usually provide poor descriptions of the project cost which directly affects the quality of a Benders' cut. Then, at a later stage, the algorithm should focus more on improving the solution of the standard dual subproblem. This is to ensure the convergence to be the same as in the classical Benders' decomposition. Therefore, Oliveira

et al. (2014) suggested dynamically adjusting the value of μ such that it gradually decreases during the process of the algorithm. Recommended characteristics of a sequence of μ in the algorithm, represented by $\{\mu^{(t)}\}_{t=1,\ldots,\infty}$, are as follows:

1. $\mu^{(t)} \to 0$ as $t \to \infty$.
2. $\sum_{t=1}^{\infty} \mu^{(t)} \to \infty$.

Under the above conditions, the convergence of the algorithm is guaranteed because

$$\lim_{\mu \to 0} \pi^T(d - B\overline{y}) + \mu\pi^T(d - B\hat{y}) = \pi^T(d - B\overline{y}).$$

To generate a near-optimal maximal nondominated cut in our model, we choose the initial core point from Algorithm 5 as the positive weight vector. Let $(\overline{g}, \overline{x})$ and $(\hat{g}, \hat{x})$ be a master solution and the positive weight vector (i.e., the initial core point) in our model. The modified subproblem to generate a near-optimal maximal nondominated cut is represented below.

$$\begin{aligned}
\max \quad & z(\alpha, \beta) := \Theta(\overline{g}, \overline{x}) + \mu\Theta(\hat{g}, \hat{x}) && \text{(Maximal}(\overline{g}, \overline{x})\text{)} \\
\text{s.t.} \quad & \alpha_b^w - \beta_{bi}^w \leq c_{bi} && b, i \in B,\ w \in W \\
& \alpha_b^w \in \mathbb{R} && b \in B,\ w \in W \\
& \beta_{bi}^w \geq 0 && b, i \in B,\ w \in W
\end{aligned}$$

where $\Theta(g, x) \equiv \sum_{b \in B} \sum_{w \in W} \sum_{p \in P_b} \alpha_b^w \psi_p^w g_{bp} - \sum_{b \in B} \sum_{i \in B} \sum_{w \in W} \beta_{bi}^w x_i^w$.

Regarding a proper value of $\{\mu^{(t)}\}_{t=1,\ldots,\infty}$, we follow the suggestion of Oliveira et al. (2014) to have the satisfying characteristics that we have presented before. Note that we favour a sequence of μ which converges to 0 and also renders into a divergent series.

Given $\mu^{(t)}$ as a value of μ in iteration t for $t = 1, \ldots, \infty$, the value of μ in the next iteration is as follows.

$$\mu^{(t+1)} = \kappa^{(t+1)}\mu^{(t)}$$

where $\kappa^{(t+1)} = \frac{2}{t}$.

According to Oliveira et al. (2014), μ for the first iteration or μ^1 is from (3.35). However, it is not practical for us since our subproblem is always feasible and then a value of M is not explicit. Therefore, we choose a fixed value for μ^1 which is sufficiently small. In this way, we have implicitly selected a specific value of M.

In terms of the algorithm description, we can simply change Line 12 of Algorithm 2 to solve (Maximal$(\overline{g}, \overline{x})$) instead, where $(\overline{g}, \overline{x})$ is a master solution.

3.4.5 Multiple Optimality Cuts

Instead of generating a single optimality cut in each iteration, Santoso et al. (2005) showed the benefit of disaggregating an optimality cut for a decomposable subproblem. The authors applied Benders' decomposition on a stochastic problem for supply chain network design, where the subproblem consists of solving N independent scenarios. They created an optimality cut for each scenario, so the subproblem provides N optimality cuts per iteration. This improves the lower bound significantly. However, the master problem can grow rapidly as the algorithm progresses and directly increases the required computational time. In such a case, we believe that it is an acceptable trade-off for the improvement on the lower bound, which leads to the better convergence of the algorithm.

The dual subproblem of our model can be separated into n smaller problems, where $n = |B| \cdot |W|$. In particular, each of them is specific for each customer in each week. The dual subproblem can be represented in the following way.

$$z(\alpha, \beta) := \sum_{b \in B} \sum_{w \in W} L_b^w(\alpha_b^w, \beta_b^w) \qquad \text{(dualSPMultiAll)}$$

where $\beta_b^w \equiv (\beta_{b1}^w, \beta_{b2}^w, \ldots, \beta_{b|B|}^w) \in \mathbb{R}_+^{|B|}$ and $L_b^w(\alpha_b^w, \beta_b^w)$ is the independent problem for customer $b \in B$ in week $w \in W$ as follows:

$$L_b^w(\alpha_b^w, \beta_b^w) = \max_{(\alpha_b^w, \beta_b^w) \in X_b^w} \sum_{p \in P_b} \alpha_b^w \psi_p^w \overline{g}_{bp} - \sum_{i \in B} \beta_{bi}^w \overline{x}_i^w \qquad \text{(dualSPMultiAll}(b, w)\text{)}$$

where X_b^w is a feasible region for the corresponding independent problem, i.e.,

$$X_b^w = \{(\alpha_b^w, \beta_b^w) : \alpha_b^w - \beta_{bi}^w \leq c_{bi},\ \alpha_b^w \in \mathbb{R},\ \beta_{bi}^w \geq 0,\ i \in B\}.$$

We derive an optimality cut from the independent problem (dualSPMultiAll(b, w)) and there are n optimality cuts per iteration.

We refine the objective function of (MP) and the optimality cuts for the multi-cut version. The other constraints for the integer variables are similar to those of the original one. The modified master problem is shown below.

$$\begin{aligned}
\min \quad & \sum_{b \in B} \sum_{w \in W} \eta_b^w && \text{(MPMultiCutAll)} \\
\text{s.t.} \quad & (3.6) - (3.9) \\
& (3.11) - (3.12) \\
& \sum_{p \in P_b} \alpha_b^w \psi_p^w g_{bp} - \sum_{i \in B} \beta_{bi}^w x_i^w \leq \eta_b^w && (\alpha_b^w, \beta_b^w) \in \pi_{\text{points}}^{SP} \qquad (3.36) \\
& \eta_b^w \geq 0.
\end{aligned}$$

3.4.6 Branch-and-Benders' Cut

Naoum-Sawaya & Elhedhli (2013) and many researchers (e.g., Fortz & Poss (2009), de Sá et al. (2013), Botton et al. (2013), Adulyasak et al. (2015), de Sá et al. (2018)) agreed that one of the main drawbacks of the classical Benders' algorithm is solving an integer programme in the master problem from scratch repeatedly. It takes an incredible amount of time as the algorithm progresses, due to the increasing number of constraints from optimality cuts and feasibility cuts. The *branch-and-Benders' cut* technique is used for alleviating this problem. This approach starts from a single branch-and-bound tree for the master problem. Benders (1962) and McDaniel & Devine (1977) asserted that Benders' cuts are global in a branch-and-bound tree, i.e., Benders' cuts in each node are valid for every node of the tree and the original problem. A formal proof can be found in Naoum-Sawaya & Elhedhli (2013). Therefore, we create a cut pool to collect all Benders' cuts from any previous nodes in the tree and provide this information to the new child nodes as a warm-start. Here, we choose to generate multiple optimality cuts (3.36) and they are Pareto-optimal cuts from Papadakos (2008) to guarantee the strength of the cuts.

Moreover, adapting the idea of two-phase Benders' decomposition, we add Benders' cuts derived from fractional solutions to strengthen the linear programming relaxation of the master problem at the root node, as in the first phase. However, too many cuts from fractional solutions can become a computational burden instead. Therefore, we limit the number of those Benders' cuts in the root node. We define the stopping criteria of adding cuts similar to the classical two-phase version: the improvement on the objective value at the node and the maximum number of iterations to add cuts. After that, we branch the root node, and whenever we find an integer solution in a child node, we generate the corresponding optimality cuts and add them to the cut pool. The stopping criteria for the algorithm are either reaching the maximum computational time or no more non-fathomed nodes to explore.

Algorithms 7 and 8 show the warm-start at the root node and the search in the tree after branching the root node, respectively. $maxTime$ represents the limitation of the computational time, which is one of the stopping criteria for both algorithms.

In Algorithm 7, the root node has additional parameters to control the number of cuts as follows: ϵ^r and N^r are the threshold of the improvement on the objective value and the maximum number of iterations to add cuts, respectively. Also, we define the necessary variables

for the algorithm as follows. Let t be the counter for the number of iterations to add cuts at a node. Let P_{Imp} be the relative percentage gap of the improvement between the objective values from two consecutive iterations. $Time$ and P represent the current time and the cut pool for the tree, respectively.

We begin Algorithm 7 with the LP relaxation of the master problem at the root node. In the case of Pareto-optimal cuts by Papadakos (2008), we exploit the independent Magnanti-Wong problem and the decomposable subproblem to add multiple Pareto-optimal cuts from an initial core point beforehand: see Lines 3−4. Line 6 shows the conditions to generate new valid cuts at the node. If we pass the criteria for controlling the number of cuts and the limitation of computational time, we generate multiple nondominated optimality cuts for a fractional solution of the LP relaxation of the master problem in the next iteration. Otherwise, we branch the root node and continue the process in Algorithm 8. Note that if the LP relaxation of the master problem at the root node is infeasible, it implies that the original problem is also infeasible. In that case, we stop the algorithm: see Line 9.

The tree search after branching the root node is shown in Algorithm 8. We introduce a new variable UB to record the best objective value of the incumbent in the search. Here, we generate multiple nondominated optimality cuts only from integer solutions. We fathom a node when one of these situations happens: the node provides an integer solution, the LP relaxation of the master problem at the node is infeasible, or the objective value of the LP relaxation of the node is worse than the incumbent value: see Lines 9 and 11, respectively. Otherwise, we branch the node. The algorithm continues until there are no more non-fathomed nodes to explore or we reach the maximum time.

Algorithm 7 Branch-and-Benders' Cut − Part 1 : Strengthening the root node

Parameter: $maxTime$, ϵ^r, N^r
Initial:
t:=0, $P_{Imp} := 100$, $Time := 0$, $P := \emptyset$,
1: Start with the LP relaxation of the master problem (MP) called (MPRelax)
2: **if** We select Pareto-optimal cuts by Papadakos (2008) for optimality cuts **then**
3: Derive an initial core point and generate multiple Pareto-optimal cuts (3.36)
4: Add the cuts to (MPRelax) at the node and to the cut pool P
5: **end if**
6: **while** $t < N^r$, $P_{Imp} > \epsilon^r$, **and** $Time < maxTime$ **do**
7: Solve (MPRelax)
8: **if** (MPRelax) is infeasible **then**
9: (OriginalSchedule) is infeasible. **Stop the algorithm**
10: **else**
11: $t := t + 1$
12: Get the optimal solution of (MPRelax)
13: Generate multiple nondominated optimality cuts (3.36)
14: Add the cuts to (MPRelax) at the node and to the cut pool P
15: Update P_{Imp} from (3.14)
16: **end if**
17: Update the current time in $Time$
18: **end while**
19: Branching the node and continue in Algorithm 8

3.5 Data Generation and Programme Set-Up

To test the efficiency of our proposed Benders' algorithm, we run the experiment on the same small data instances in the same computer from Section 2.5.

According to the experiment in Section 2.5, the combinations with more customers and longer planning horizon, for example, 30_3, 40_3, and 50_3 are significantly harder to solve to optimality. The parameter value of τ^{week} is 0.05 which is the same.

Algorithm 8 Branch-and-Benders' Cut − Part 2 : Tree search

Parameter: $maxTime$
Initial:
$UB := 1e10$
while There is non-fathomed node **and** $Time < maxTime$ **do**
  Select a non-fathomed node from the tree
  Solve (MPRelax) at the node
  Get the optimal solution of (MPRelax), (g, x), with the objective value η
  if (g, x) is integer **then**
    Generate the corresponding multiple nondominated optimality cuts (3.36)
    Add the cuts to (MPRelax) at the node and to the cut pool P
    Update $UB := \min\{UB, \eta\}$
    Fathom the node
  else if $\eta > UB$ **or** (MPRelax) is infeasible **then**
    Fathom the node
  else
    Branching the node
  end if
  Update the current time in $Time$
end while

The experiment is run on a single thread with no pre-processing. The satisfactory tolerance level between the bounds in the tree is 0.01%, which is the default value for CPLEX. The maximum computational time for each data instance is 900 seconds (15 minutes).

Regarding a nondominated optimality cut, we focus on a Pareto-optimal cut from Papadakos (2008) and a maximal nondominated cut from Sherali & Lunday (2013). For a Pareto-optimal cut, we set the number of feasible solutions to generate an initial core point (N^g) in Algorithm 5 at 10. For a maximal nondominated cut, we use the initial core point as the positive weight vector and select 10^{-6} for the initial value of μ.

The values of the parameters that control the number of Benders' cuts from fractional solutions at the root node in Algorithm 7, ϵ^r and N^r, are 0.5% and 100, respectively. Concerning the location-allocation heuristic, we can solve the location problem quickly by hand, so we use CPLEX to solve only the mixed-integer programme of the allocation problem. We set 10 seconds as the maximum time to solve the allocation problem.

To set up the branch-and-Benders' cut in CPLEX, we use the callback classes: *user cut callback* and *lazy constraint callback*. According to *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual* (2017), user cuts are cuts defined by a user to tighten the feasible region of the LP relaxation and do not cut off any feasible integer solutions. Note that the user cut callback can be activated for generating cuts at every node except when a feasible integer solution is found. In contrast, lazy constraints are used for checking violations of Benders' optimality cuts for integer solutions. Therefore, the user cut callback is invoked only at the root node for adding Benders' cuts from fractional solutions, while the lazy constraint callback is activated for every integer solution found in the tree.

3.6 Computational Results

First, we compare the performance of a manual optimality cut from (3.26) and a nondominated optimality cut from Papadakos (2008) and Sherali & Lunday (2013) in the form of a single optimality cut (3.10). This is to show the strength of a nondominated optimality cut. Then, we show the advantages of using multiple optimality cuts and a high-quality initial integer solution. Also, we investigate different cut strategies on the branch-and-Benders' cut. Finally, we compare our best version of the algorithm to the modern branch-and-cut and the Benders' decomposition algorithm in CPLEX.

Data	Best %Gap			%Gap			Worst %Gap		
	Manual	**Papa**	**Maxi**	**Manual**	**Papa**	**Maxi**	**Manual**	**Papa**	**Maxi**
30_1	0.01	0.01	0.01	8.08	7.90	8.93	15.38	14.87	16.71
30_2	13.06	12.15	13.63	25.81	25.38	26.23	38.23	38.82	38.29
30_3	53.53	50.94	57.31	64.64	61.15	65.22	84.71	80.86	84.88
40_1	2.01	1.95	2.54	14.11	12.34	14.70	21.86	17.95	24.85
40_2	14.71	14.62	14.66	24.85	24.79	25.40	36.09	36.92	36.10
40_3	58.80	57.27	61.60	67.32	64.93	68.79	79.99	76.88	81.92
50_1	2.83	2.81	4.13	12.91	12.64	14.54	16.77	16.86	19.06
50_2	26.31	25.05	28.63	28.97	27.14	29.64	33.56	31.92	32.95
50_3	61.25	60.41	62.14	69.23	66.74	68.99	77.76	73.56	77.80

Table 3.1: The performance of the manual and nondominated optimality cuts in the form of a single cut (1).

3.6.1 Comparison between Different Nondominated Optimality Cuts

Table 3.1 and 3.2 show the comparison of the overall results between a manual optimality cut (3.26) and a nondominated optimality cut from Papadakos (2008) and Sherali & Lunday (2013), where they are in the form of a single optimality cut. *Manual*, *Papa*, and *Maxi*, respectively, represent the manual optimality cut from Section 3.4.3, a Pareto-optimal cut from Papadakos (2008), and a maximal nondominated cut from Sherali & Lunday (2013).

Similarly to Section 2.5, we focus on the solution quality and the average computational time in seconds. Table 3.1 shows the information of the relative percentage gaps where *Best %Gap*, *%Gap*, *Worst %Gap* are the best, average, and the worst value of the relative percentage gaps among five data instances, respectively. Overall, *Papa* performs best, especially in the most difficult combinations such as 30_3, 40_3, and 50_3. *Manual* is the second-best, while *Maxi* comes last. Additionally, we show the number of data instances that can find the optimal solution within the limitation of time in Columns *#Opt (45)* of Table 3.2, where the number in brackets is the total number of data instances. The last row of the table, *Total*, presents the total of *#Opt* from every combination for each algorithm to show the overall results. There is, however, no difference in the number of *#Opt* as every method can find the optimal solution in a few instances of 30_1 which is the least challenging combination.

In terms of the average computational time, Column *Total time (s)* of Table 3.2 shows that there is no significant difference among the three techniques.

From the above results, *Papa* has the best overall performance. It is also interesting that *Manual* tends to perform slightly better than *Maxi* although it does not have any guarantee on the cuts' strength. We suspect that we do not exploit the main strength of a maximal nondominated cut, since we can find an initial core point easily and the subproblem is quick to solve. In particular, such characteristics reinforce a Pareto-optimal cut as shown in the further investigation below.

Further Investigation on a Pareto-optimal Cut by Papadakos (2008)

To see the real advantages of the Pareto-optimal cuts, we investigate further the step-by-step change of the objective value at the root node. CPLEX updates the objective value after any cut has been added at the root node. In the case of *Papa*, the first objective value shows the effect of an initial cut from Papadakos (2008), which we add before starting the branch-and-Benders' cut. It should be noted that apart from Benders' cuts from fractional solutions added by the user cut callback, CPLEX provides additional cuts, for example, cover cuts, zero-half cuts, and Gomory fractional cuts, for further tightening the LP relaxation (*IBM ILOG CPLEX Optimization Studio CPLEX User's Manual* 2017). Here, we let CPLEX add all of its cuts before our user cuts. Moreover, CPLEX uses its heuristic to generate an integer solution. In that case, the lazy constraint callback is evoked to create a Benders' cut.

We select some representative data instances and then plot a graph for each of them to

Data	#Opt (45)			Total time (s)		
	Manual	**Papa**	**Maxi**	**Manual**	**Papa**	**Maxi**
30_1	2	2	2	554.8	553.8	576.6
30_2	0	0	0	900	899.8	901.4
30_3	0	0	0	900	906.4	912.8
40_1	0	0	0	900	900.2	900.6
40_2	0	0	0	900	900.2	901.4
40_3	0	0	0	900	903	907.2
50_1	0	0	0	900	900	901
50_2	0	0	0	900	900.6	901.4
50_3	0	0	0	900	902.8	904.4
Total	2	2	2			

Table 3.2: The performance of the manual and nondomainated optimality cuts in the form of a single cut (2).

present the change of its objective value at the root node. Here, we consider the relative percentage gap between the objective value and the best-found integer solution of the corresponding data instances (found by either *Manual*, *Papa*, or *Maxi*) for a fair comparison. Figures 3.1a−3.1e show the trend of those relative percentage gaps of the instances for the combinations 30_1, 30_2, 40_1, 40_2, and 50_2 respectively. To separate the objective value that is improved by the CPLEX cuts, its symbol on the graph is not filled with colour. Every figure shows that after CPLEX adds all its cuts, our cuts are capable of further improving the objective value. Moreover, for these data instances the initial cut from *Papa* can improve the objective value immediately: see the first relative percentage gap in each figure.

In more difficult data instances, the initial cut from *Papa*, however, does not always improve the objective value at the beginning. Nevertheless, *Papa* manages to reach smaller relative percentage gaps more quickly before branching the root node, as shown in Figure 3.2. Note that the main reduction in the relative percentage gap of the bounds happens at the root node. Therefore, a smaller relative percentage gap in the root node tends to achieve better convergence of the bounds.

Also, we test the effect of a different choice of an initial core point. In addition to the initial core point from Algorithm 5, we generate an initial (approximate) core point from a solution of the LP relaxation of the master problem. This approximation of core points can be found in Santoso et al. (2005). The results of Pareto-optimal cuts from Papadakos (2008) with different initial core points are shown in Table 3.3. *LPR* and *Feasible* represent those where an initial core point is generated from the LP relaxation of the master problem and Algorithm 5, respectively.

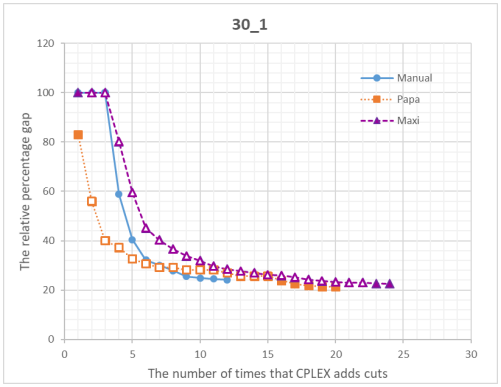
Although the two methods do not have different results in the number of *#Opt*, *Feasible* shows superior performance in terms of the average and the best relative percentage gap in every combination of the data instances. Regarding *Worst %Gap*, *Feasible* clearly has better performance than those of *LPR* when the data instances are more difficult to solve. This is not surprising since the initial (approximate) core point from the LP relaxation is more likely to not be a real core point. As a result, the point might require some time to be updated by Equations (3.30) and (3.31), before actually converging to a real core point to generate a real Pareto-optimal cut. This results in a slower convergence. In terms of the average time, both algorithms are similar.

The results confirm that the choice of an initial core point affects the overall performance of the Pareto-optimal cuts. Also, Algorithm 5 provides a good initial core point for generating the Pareto-optimal cuts.

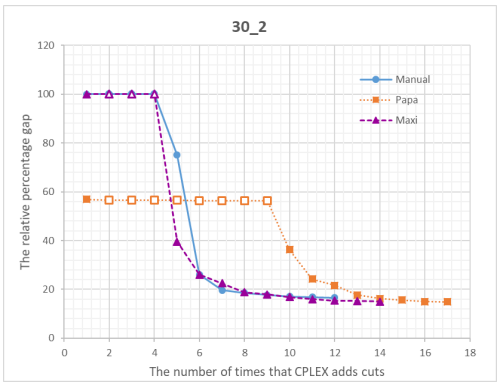
The further investigation provides more insightful information to understand the best performance of the Pareto-optimal cuts. In particular, it shows that the Pareto-optimal cuts can improve the objective values at the root node immediately or derive better objective values be-

Data	Best %Gap		%Gap		Worst %Gap		#Opt (45)		Total time (s)	
	LPR	**Feasible**	**LPR**	**Feasible**	**LPR**	**Feasible**	**LPR**	**Feasible**	**LPR**	**Feasible**
30_1	0.01	0.01	8.16	7.90	14.68	14.87	2	2	555.6	553.8
30_2	13.08	12.15	25.62	25.38	37.39	38.82	0	0	900	899.8
30_3	54.53	50.94	62.91	61.15	80.11	80.86	0	0	900	906.4
40_1	2.58	1.95	14.01	12.34	20.73	17.95	0	0	900	900.2
40_2	15.51	14.62	25.54	24.79	36.01	36.92	0	0	900	900.2
40_3	60.40	57.27	67.00	64.93	77.20	76.88	0	0	900	903
50_1	2.86	2.81	13.56	12.64	17.47	16.86	0	0	900	900
50_2	26.88	25.05	28.23	27.14	32.29	31.92	0	0	900	900.6
50_3	62.45	60.41	67.95	66.74	76.40	73.56	0	0	900	902.8
Total							2	2		

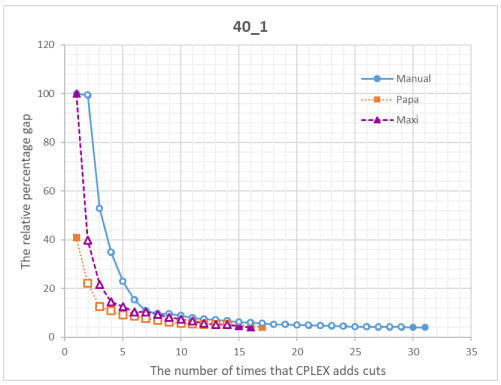
Table 3.3: The performance of a different initial core point for the Pareto-optimal cuts.



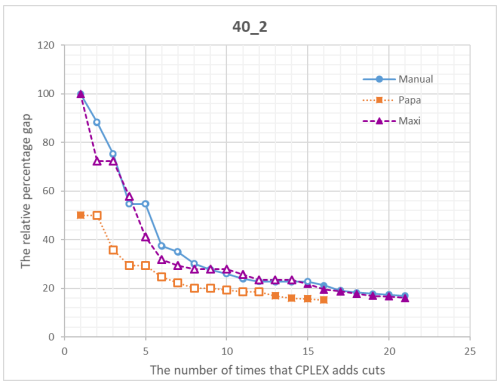
(a) Data instance for 30_1.



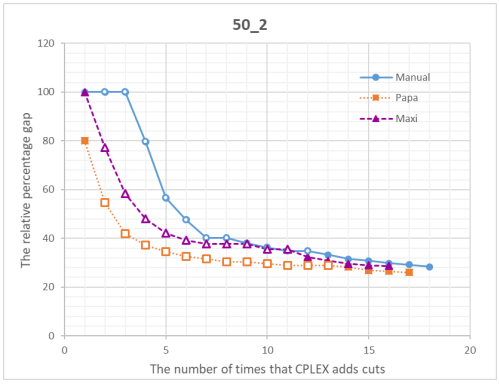
(b) Data instance for 30_2.



(c) Data instance for 40_1.



(d) Data instance for 40_2.



(e) Data instance for 50_2.

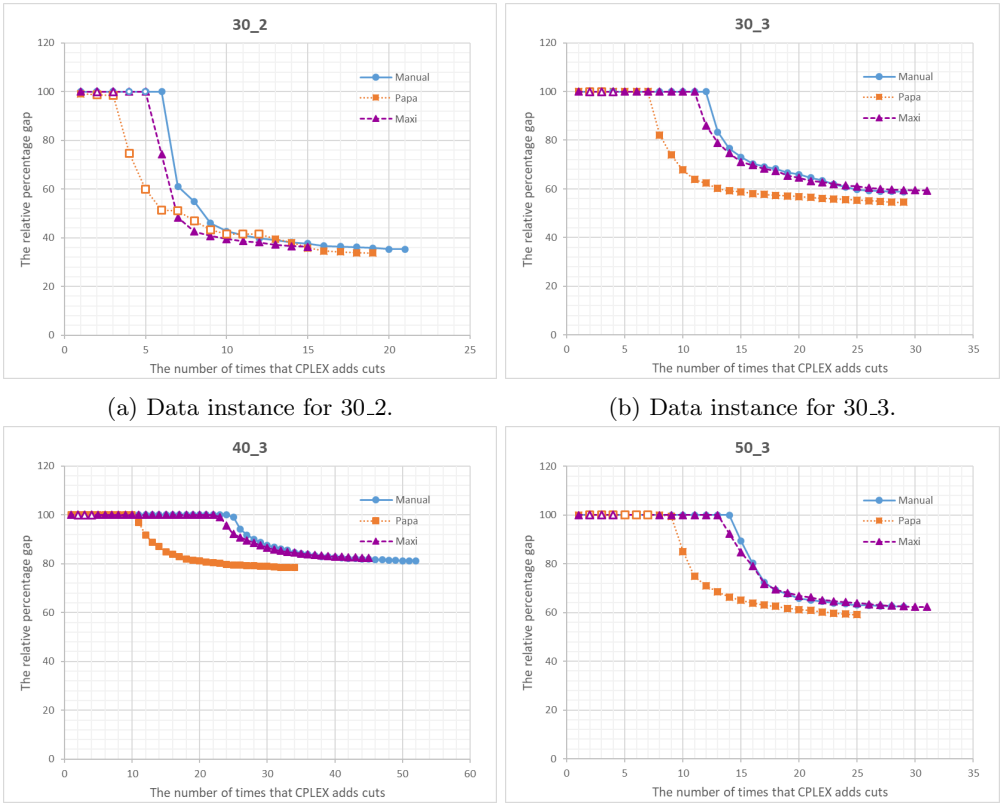
Figure 3.1: The instances that an initial cut from *Papadakos* improves the objective value of the root node immediately.

fore branching the root node. The main reason for the algorithm efficiency is the high-quality initial core point that is easy to derive.

Due to the advantages of the Pareto-optimal cuts that we have discussed, we conclude that a Pareto-optimal cut is the most suitable nondominated optimality cut for our model. We then select it as an optimality cut in the remainder. However, *Papa* still struggles to solve the problem as they cannot find the optimal solutions in other combinations but 30_1 within the limitation of time. Therefore, more accelerating techniques are still necessary for the algorithm.

3.6.2 Advantages of Multiple Optimality Cuts

Next, we focus on the performance of a Pareto-optimal cut in the form of the single optimality cut and the multiple cuts. We showed in Section 3.4.5 that the subproblem can be decomposed



(a) Data instance for 30_2.

(b) Data instance for 30_3.

(c) Data instance for 40_3.

(d) Data instance for 50_3.

Figure 3.2: The instances that *Papadakos* reaches a better relative percentage gap before branching the root node.

into a smaller problem for each customer and each week. To study the effect of a strategy to decompose the subproblem, we also consider the multiple cuts from the partially decomposed subproblem. Namely, we decompose the subproblem for only one of each week or each customer. Each optimality cut, then, for the former case and the latter case, is represented for each week and each customer, respectively. We generate the optimality cuts for those cases by partially aggregating the multiple optimality cuts (3.36). To represent an optimality cut for each week, we aggregate the optimality cuts of all customers in that week, shown in (3.37). Similarly, (3.38) is an optimality cut for a particular customer created by aggregating the corresponding optimality cuts from every week.

$$\sum_{b\in B}\left(\sum_{p\in P_b}\alpha_b^w\psi_p^w g_{bp}-\sum_{i\in B}\beta_{bi}^w x_i^w\right)\le\eta^w \qquad (\alpha_b^w,\beta_b^w)\in\pi^{SP}_{\text{points}} \tag{3.37}$$

$$\sum_{w\in W}\left(\sum_{p\in P_b}\alpha_b^w\psi_p^w g_{bp}-\sum_{i\in B}\beta_{bi}^w x_i^w\right)\le\eta^b \qquad (\alpha_b^w,\beta_b^w)\in\pi^{SP}_{\text{points}} \tag{3.38}$$

where $\eta^w\equiv\sum_{b\in B}\eta_b^w$ and $\eta^b\equiv\sum_{w\in W}\eta_b^w$.

The comparison of the results for the different forms of the Pareto-optimal cuts is in Tables 3.4 and 3.5. *Single*, *Week*, *Cus*, and *All* denote the single optimality cut (3.10), the partially aggregated cuts (3.37), (3.38), and the completely disaggregated cuts (3.36), respectively.

In terms of relative percentage gaps, Table 3.4 shows that every form of the multiple cuts improves upon the performance of the single cut. In particular, *All*, *Cus*, and *Week* perform the best, the second best and the third-best, respectively. Clearly, *All* provides outstanding results. It improves the average relative percentage gap of *Single* by more than 91% in all cases except

in the most complicated combinations, such as 40_3 and 50_3. The improvements for those are 74% and 69%, respectively, which are still considerable amounts. The outperformance of *All* is also supported by the information of *Best %Gap* and *Worst %Gap* in every combination. Moreover, *All* has significantly more instances that find the optimal solution, especially in the more complicated combinations like 30_2, 30_3, and 40_2: see Columns *#Opt (45)* in Table 3.5. As a result, it has the highest total of *#Opt*, which is 22 instances more than that of *Single*.

Regarding the computational time in Table 3.5, for less complicated combinations, e.g., 30_1, 30_2, 40_1, and 50_1, an increasing number of Benders' cuts per iteration tends to reduce the computational time on average. *All* considerably outperforms the other algorithms in almost all of these combinations. For example, compared to *Single*, it reduces the time massively in 50_1 by more than one order of magnitude. However, it does not perform best in 30_1. In such combination, *Cus* spends the least amount of time which is around 32% less time on average to reach the same average relative percentage gaps as *All*. However, *All* spends only 40 seconds on average which is still an acceptable amount of time. Therefore, *All* is still outstanding in terms of computational time.

Overall, *All* performs best. This implies that a larger number of cuts per iteration results in significantly improved performance. Although there is a risk of adding too many cuts at the root node, our stopping criteria of adding cuts, especially the relative percentage gap on the improvement, manage to avoid that problem.

Multiple Cuts Improve the Efficiency of the Initial Cuts

Further investigation shows that the multiple cuts can improve the efficiency of the initial cuts from Papadakos (2008). We choose some data instances and plot graphs in a similar way to Figures 3.1 and 3.2 for *Single*, *Week*, *Cus*, and *All*. Namely, we focus on the change in the relative percentage gap between the objective values at the root node and the best-found integer solution value for each of those data instances. The best found integer solution here is from the best one among *Single*, *Week*, *Cus*, and *All* for any corresponding data instance. Also, we separate the objective value that is improved by CPLEX's cuts; It is represented by a symbol without colour inside.

Figure 3.3 shows the graphs of the change in the relative percentage gap at the root node for the data instances from the more challenging combinations, such as 30_3, 40_3, 50_2, and 50_3. In these data instances, the initial cut from Papadakos (2008) in the completely aggregated form struggles to improve the objective value at the beginning: see the graphs of *Single* in Figures 3.3a−3.3d. *Week* and *Cus* face almost the same situation for most of the data instances. However, Figure 3.3c shows that the initial cuts in the partially aggregated form can improve the objective values at the beginning. It is clear that the initial cuts from *All* are helpful and reduce the relative percentage gap immediately. Therefore, the completely disaggregated form for the Pareto-optimal cuts enhances the quality of initial cuts the most, thereby providing an excellent start for the algorithm.

3.6.3 Benefit of the Initial Integer Solution

We now study the benefit of an initial integer solution from Section 3.4.2 that is used to generate cuts and set the upper bound of the problem before the branch-and-Benders' cut. Table 3.6 presents the comparison of the performance of the completely disaggregated Pareto-optimal cuts without and with the initial integer solution to start, where *NoHeu* and *Heu* represent the former and the latter, respectively.

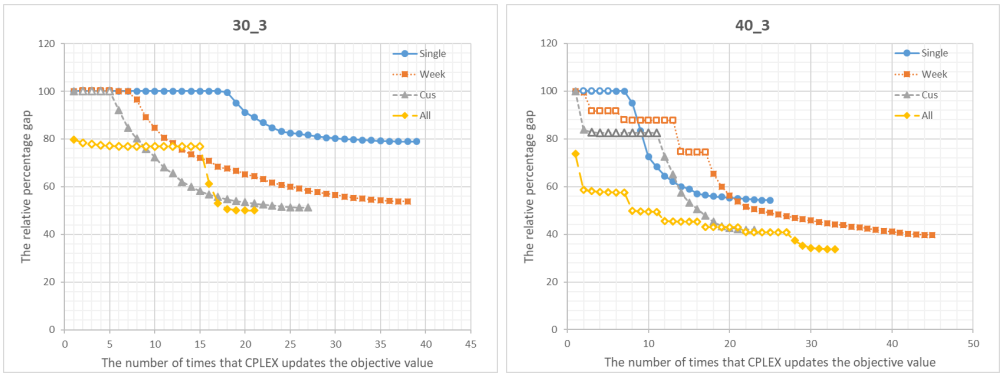
In terms of the average relative percentage gap, both algorithms are quite competitive in most combinations except the most challenging combinations, such as 40_3 and 50_3. For these two complicated combinations, *Heu* shows a better performance. This is also consistent with the corresponding information of *Best %Gap* and *Worst %Gap*. Although *NoHeu* has smaller average relative percentage gaps in less difficult combinations, like 30_3, 40_2, and 50_2, those of *Heu* are not significantly different. Regarding the number of instances which found the optimal solution, *NoHeu* gets more of those instances in a few combinations, for example, 30_3, and 40_2, but overall *Heu* is still competitive with it: see the total of *#Opt*.

Data	Best %Gap				%Gap				Worst %Gap			
	Single	**Week**	**Cus**	**All**	**Single**	**Week**	**Cus**	**All**	**Single**	**Week**	**Cus**	**All**
30_1	0.01	0.01	0.00	0.00	7.90	1.63	0.01	0.01	14.87	4.71	0.01	0.01
30_2	12.15	3.85	0.01	0.01	25.38	17.81	4.47	0.79	38.82	29.14	12.55	3.91
30_3	50.94	33.48	11.73	0.01	61.15	41.06	18.38	3.97	80.86	54.21	31.38	13.38
40_1	1.95	0.01	0.01	0.01	12.34	3.82	0.17	0.05	17.95	9.57	0.82	0.24
40_2	14.62	8.09	3.76	0.01	24.79	17.40	5.55	0.07	36.92	28.39	7.67	0.30
40_3	57.27	44.48	22.49	5.76	64.93	51.75	28.65	16.83	76.88	60.33	34.79	24.16
50_1	2.81	0.01	0.01	0.01	12.64	3.15	0.01	0.01	16.86	5.69	0.01	0.01
50_2	25.05	17.24	5.51	0.69	27.14	21.72	7.97	2.29	31.92	25.81	10.31	4.62
50_3	60.41	50.46	27.76	16.71	66.74	56.19	31.00	20.79	73.56	60.12	35.86	23.44

Table 3.4: The performance of the Pareto-optimal cuts in the form of a single cut and multiple cuts (1).

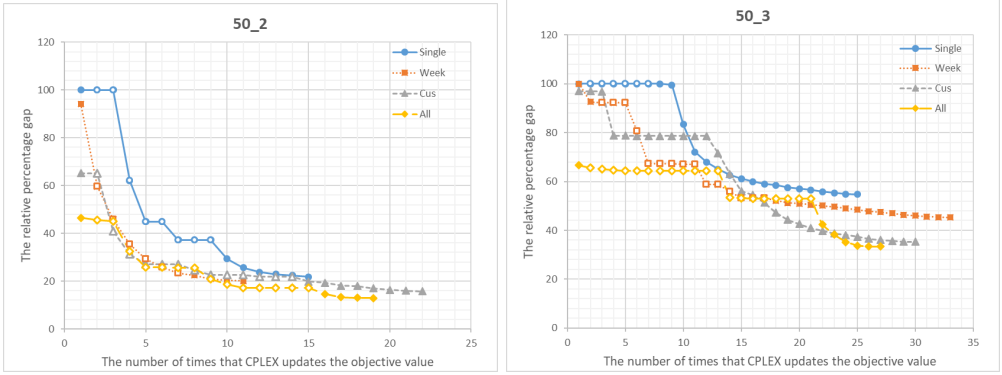
Data	#Opt (45)				Total time (s)			
	Single	**Week**	**Cus**	**All**	**Single**	**Week**	**Cus**	**All**
30_1	2	3	5	5	553.8	484.2	27.4	40.4
30_2	0	0	1	4	899.8	901.6	776.8	316.4
30_3	0	0	0	2	906.4	909.8	906.6	817.8
40_1	0	1	4	4	900.2	723.2	308.6	234.4
40_2	0	0	0	4	900.2	902	900.6	640.6
40_3	0	0	0	0	903	905.6	902.8	903.6
50_1	0	1	5	5	900	737.6	291.2	81.6
50_2	0	0	0	0	900.6	900.8	900.2	902
50_3	0	0	0	0	902.8	902.2	902	902.2
Total	2	5	15	24				

Table 3.5: The performance of the Pareto-optimal cuts in the form of a single cut and multiple cuts (2).



(a) Data instance for 30_3.

(b) Data instance for 40_3.



(c) Data instance for 50_2.

(d) Data instance for 50_3.

Figure 3.3: The instances that the multiple cuts enhance the initial cuts from *Papadakos*.

In terms of the average execution time, *Heu* shows between 5% and 81% of improvement in all combinations that do not reach the maximum time, except 40_2. In that combination, the computational time by *Heu* is around 14% higher than that of *NoHeu*. However, this is still an acceptable amount.

In conclusion, the results show that the initial integer solution reduces the average computational time to reach similar relative percentage gaps to those of *NoHeu*. This is because the location-allocation heuristic generates a high-quality initial integer solution whose objective value helps to prune more nodes during the search. Therefore, the initial integer solution

improves the performance, especially for more challenging data instances.

3.6.4 Further Investigation on Cut Strategies

Different Cut Strategies for Benders' Cuts from Fractional Solutions

According to Botton et al. (2013), a recommended cut strategy for the Benders' cuts from fractional solutions is adding them only at the root node, since adding those cuts at every node is not effective. Here, we will check the advantages of the recommended cut strategy by comparing it to the one that adds those cuts at every node. For adding those cuts at every node, we modify Algorithm 8 by also allowing for Benders' cuts from fractional solutions to be added in each node. These cuts aim at strengthening the LP relaxation of the node, similar to those at the root node. We apply the identical stopping criteria for adding the cuts at any child node: the improvement on the objective value at the node and the maximum number of iterations to add the cuts. The stopping criteria for the child nodes are stricter to avoid generating too many cuts in the tree. The threshold of the improvement on the objective value is 1% and the maximum number of iterations to add cuts is 5.

We set up the experiment on the completely disaggregated Pareto-optimal cuts with an initial integer solution. The results of the different cut strategies are shown in Table 3.7, where *OR* and *E* stand for adding Benders' cuts from fractional solutions at only the root node and every node, respectively.

OR has the overall best performance in the solution quality, especially when data instances become more complicated. Namely, *OR* has superior average percentage gaps. Also, the best relative percentage gap for every combination of *OR* is better than that of *E*: see Column *Best %Gap*. Furthermore, the total of *#Opt* shows that overall *OR* manages to find the optimal solutions in more data instances.

In terms of the computational time, *OR* outperforms *E* in almost every combination. *OR* requires a significantly smaller amount of computational time in 30_1, 40_1, and 50_1. Also, *OR* can complete the experiment before reaching the maximum time in 30_3, while *E* spends the full time and still has the worst relative percentage gap on average. The only combination that *OR* perform worse than *E* is 40_2. However, it spends more time by only 15% which is not a considerable amount.

In conclusion, the above observation confirms that the cut strategy of adding Benders' cuts from fractional solutions only at the root node is more effective.

Different Parameter Values for the Stopping Criterion at the Root Node

The threshold of the relative percentage gap of the improvement at the root node, ϵ^r, is the main stopping criterion that controls the number of Benders' cuts from fractional solutions added at the root node. We have shown so far that the algorithm performs well with a parameter value of 0.5%. Now, we will test the effect of a different range of values for this parameter. The additional values that we will compare the results with are 0.05% and 5%. We keep the values small since Botton et al. (2013) suggested adding as many cuts as possible at the root node.

Table 3.8 shows the performance of the different values of parameter ϵ^r at the root node for the completely disaggregated Pareto-optimal cuts with the initial integer solution.

Regarding the relative percentage gaps, overall *0.5%* performs best in most combinations, including the more challenging ones such as 40_3, 50_2, and 50_3. Although it does not perform best in some aspects, for example, in terms of the best percentage gap in 40_3, it is still competitive with the best ones. In terms of the total of *#Opt*, there is no significant difference in any algorithm.

Regarding the average time, we focus on only the combinations that do not reach the limitation of time, e.g., every combination except 40_3, 50_2, and 50_3. Overall, *0.5%* still performs well. It has the best performance in 30_2, 40_1, and 50_1. For the rest of these combinations, *0.5%* does not spend the least average computational time but its performance is still comparable to the best ones. In 30_1, *0.5%* spends almost the same time as *5%*. For 30_3 and 40_2, *0.5%* spends more time than the best one (*0.05%*), but by only 12% and 1.5%, respectively, which are still acceptable amounts.

Data	Best %Gap		%Gap		Worst %Gap		#Opt (45)		Total time (s)	
	NoHeu	**Heu**	**NoHeu**	**Heu**	**NoHeu**	**Heu**	**NoHeu**	**Heu**	**NoHeu**	**Heu**
30_1	0.00	0.00	0.01	0.01	0.01	0.01	5	5	40.4	7.6
30_2	0.01	0.01	0.79	0.01	3.91	0.01	4	5	316.4	261
30_3	0.01	0.01	3.97	4.92	13.38	15.43	2	1	817.8	777.8
40_1	0.01	0.01	0.05	0.01	0.24	0.01	4	5	234.4	102
40_2	0.01	0.01	0.07	0.49	0.30	0.95	4	2	640.6	727.2
40_3	5.76	4.12	16.83	14.21	24.16	19.09	0	0	903.6	904
50_1	0.01	0.01	0.01	0.01	0.01	0.01	5	5	81.6	66.4
50_2	0.69	1.04	2.29	2.73	4.62	4.89	0	0	902	901.8
50_3	16.71	13.48	20.79	17.60	23.44	21.10	0	0	902.2	902.6
Total							24	23		

Table 3.6: The performance of multiple Pareto-optimal cuts with and without an initial integer solution.

Data	Best %Gap		%Gap		Worst %Gap		#Opt (45)		Total time (s)	
	OR	E	OR	E	OR	E	OR	E	OR	E
30_1	0.00	0.00	0.01	0.01	0.01	0.01	5	5	7.6	70.4
30_2	0.01	0.01	0.01	0.13	0.01	0.63	5	4	261	377.4
30_3	0.01	0.30	4.92	5.66	15.43	13.60	1	0	777.8	908
40_1	0.01	0.01	0.01	0.14	0.01	0.66	5	4	102	348
40_2	0.01	0.01	0.49	0.30	0.95	1.00	2	3	727.2	619.4
40_3	4.12	5.08	14.21	15.92	19.09	24.73	0	0	904	903.4
50_1	0.01	0.01	0.01	0.01	0.01	0.01	5	5	66.4	196.6
50_2	1.04	1.86	2.73	3.76	4.89	5.85	0	0	901.8	901.4
50_3	13.48	15.04	17.60	18.06	21.10	20.60	0	0	902.6	901.8
Total							23	21		

Table 3.7: The performance of the different cut strategies.

From the investigation of the solution quality and average computational time, the original value of ϵ^r is still appropriate for our algorithm.

3.6.5 Comparison between Developed Benders' Algorithm and CPLEX

Summing up, the best version of our branch-and-Benders' cut is using the multiple Pareto-optimal cuts (3.36) with an initial integer solution from the location-allocation heuristic. Finally, we compare its performance to the modern branch-and-cut and the Benders' decomposition algorithm in CPLEX. Tables 3.9 and 3.10 show the performance of CPLEX and our best version. *Default* and *AutoBD* represent the results of the modern branch-and-cut and the Benders' decomposition algorithm from CPLEX, while *Papa* is from our proposed branch-and-Benders' cut. Note that the results of *Default* were shown before in Section 2.5.

Concerning the relative percentage gaps in Table 3.9, there is no significantly difference among the algorithms for the less challenging combinations, e.g., 30_1, 30_2, 40_1, 40_2, and 50_1. When the data instances become more complicated, *AutoBD* and *Papa* reach better relative percentage gaps than *Default*. For 30_3 and 40_3, *AutoBD* reaches the smallest average relative percentage gaps; It improves the average percentage gaps of *Default* by 70% and almost 50%, respectively. *Papa* comes second with better average relative percentage gaps than those of *Default* by almost 40% in these two combinations. For 50_2 and 50_3, *AutoBD* and *B&C Papa* become competitive and improve the average relative percentage gaps of *Default* by almost 40%. This is supported by the corresponding results of *Best %Gap* and *Worst %Gap*.

Columns *#Opt (45)* in Table 3.10 supports the outstanding performances of *AutoBD*. In the combinations like 30_3 and 40_3 that the other algorithms struggle to find the optimal solutions, *AutoBD* is successful in this task, resulting in the highest total of *#Opt* shown in the last row. *Papa* manages to find the optimal solution in one data instance of such challenging combinations. However, it is not successful in finding the optimal solutions in the data instances of 40_2. As a result, it has the least total of *#Opt*.

Columns *Total time (s)* of Table 3.10 shows the average computational time to solve each combination. For 30_1, *Papa* spends the least average time. However, it is not significantly different from the others since all three methods manage to finish the algorithm fast. For the other less challenging instances, e.g., 30_2, 40_1, and 40_2, *Default* shows superior performances. In particular, *Default* spends around 55% and almost 60% less time than the other algorithms on 30_2 and 40_2, respectively. For 40_1, *Default* spends 70% and 40% less time than *Papa* and *AutoBD*, respectively. However, when the instances become more complicated like 30_3 and 40_3, *AutoBD* is the best in this aspect.

In conclusion, *Default* seems to outperform the other algorithms for the less challenging instances, e.g., 30_1, 30_2, 30_2, 40_1, and 40_2. However, *AutoBD* and *Papa* show overall superior performance for more difficult combinations. In particular, *AutoBD* and *Papa* performs best and second best, respectively.

Next, we will further investigate the performances of *AutoBD* to understand its superiority

Data	Best %Gap			%Gap			Worst %Gap			#Opt (45)			Total time (s)		
	0.05%	**0.50%**	**5%**	**0.05%**	**0.50%**	**5%**	**0.05%**	**0.50%**	**5%**	**0.05%**	**0.50%**	**5%**	**0.05%**	**0.50%**	**5%**
30_1	0.01	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01	5	5	5	10	7.6	7
30_2	0.01	0.01	0.01	0.32	0.01	0.28	1.55	0.01	1.38	4	5	4	281.6	261	265.4
30_3	0.01	0.01	0.01	4.81	4.92	4.30	18.39	15.43	17.31	2	1	2	696	777.8	787.8
40_1	0.01	0.01	0.01	0.04	0.01	0.01	0.17	0.01	0.01	4	5	5	261.4	102	142.8
40_2	0.01	0.01	0.01	0.37	0.49	0.42	1.15	0.95	0.92	3	2	2	716.8	727.2	738
40_3	3.12	4.12	7.17	14.52	14.21	15.21	24.80	19.09	22.13	0	0	0	903.8	904	904
50_1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	5	5	5	80.6	66.4	96.2
50_2	0.92	1.04	1.38	2.76	2.73	2.86	4.60	4.89	5.25	0	0	0	902	901.8	901.6
50_3	16.39	13.48	14.85	20.13	17.60	18.15	25.07	21.10	21.21	0	0	0	902.6	902.6	902.6
Total										23	23	23			

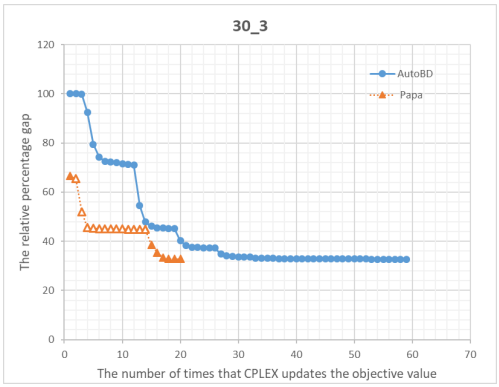
Table 3.8: The performance of the different threshold relative percentage gaps of the improvement at the root node.

over *Papa*.

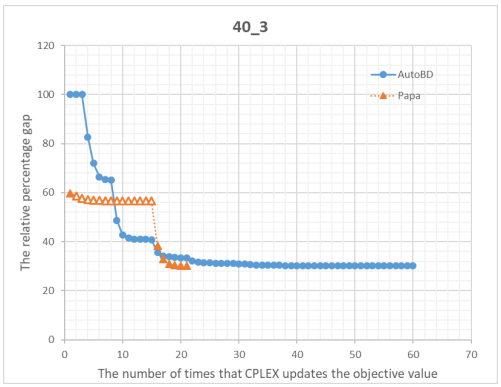
Further Investigation on *AutoBD* and *Papa*

We now further analyse the results of CPLEX and our best version to see how CPLEX derives better relative percentage gaps. First, we check the root node as it is the main place to reduce the percentage gap. Figure 3.4 represents graphs of the improvement in the relative percentage gap at the root node, similar to Figures 3.1−3.3, for the more difficult data instances, such as 30_3, 40_3, 50_2, and 50_3. In this case, the objective value of the LP relaxation at the root node is compared to the best-found integer solution by either *CPLEX*, *AutoBD*, or *Papa* for each data instance. For *Papa*, symbols without and with filling colour represent the objective values that are improved by CPLEX's cuts and by our cuts, respectively.

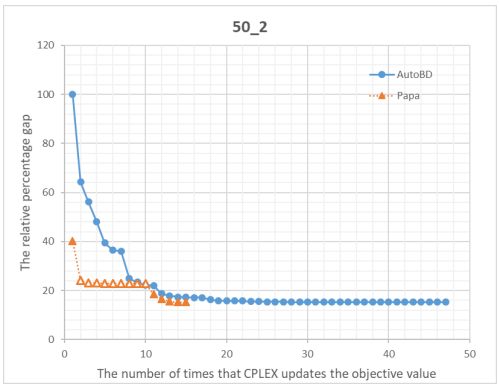
It is clear that the initial cuts from Papadakos (2008) in *Papa* reduce the relative percentage gap immediately, while there is no improvement in *AutoBD* at the beginning. Also, the graphs show that these two methods reach almost the same relative percentage gaps before branching the node, i.e., their objective values at the root node are not different at the beginning of the tree search. In fact, *Papa* spends much less computational time on average at the root node, especially for those difficult instances, as shown in Columns *TimeRN (s)* of Table 3.11. These results imply that *Papa* actually outperforms *AutoBD* at the root node. Therefore, the observation at the root node is inconclusive, i.e., it cannot explain to us why *AutoBD* derives better percentage gaps at the end.



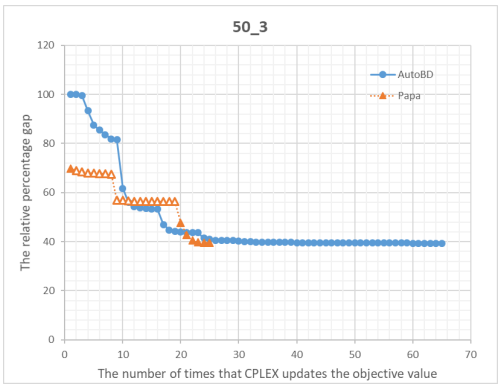
(a) Data instance for 30_3.



(b) Data instance for 40_3.



(c) Data instance for 50_2.



(d) Data instance for 50_3.

Figure 3.4: The change of relative percentage gaps at the root node in more difficult instances by *AutoBD* and *Papa*.

Since the information at the root node cannot provide any clear reason for *AutoBD*'s outstanding performances to us, we then observe the information about the upper bound and the lower bound in every data instance. The upper bound and the lower bound of each data instance are from the best-found integer solution and the best bound of the tree search, respectively.

Data	Best %Gap			%Gap			Worse %Gap		
	Default	**AutoBD**	**Papa**	**Default**	**AutoBD**	**Papa**	**Default**	**AutoBD**	**Papa**
30_1	0.00	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01
30_2	0.00	0.01	0.01	0.01	0.46	0.01	0.01	2.25	0.01
30_3	2.57	0.01	0.01	7.81	2.34	4.92	23.54	8.29	15.43
40_1	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
40_2	0.01	0.01	0.01	0.01	0.15	0.49	0.01	0.70	0.95
40_3	14.60	0.01	4.12	23.34	12.40	14.21	28.53	21.29	19.09
50_1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
50_2	3.47	1.31	1.04	4.57	2.41	2.73	6.21	4.53	4.89
50_3	23.20	11.08	13.48	32.10	17.46	17.60	41.33	21.70	21.10

Table 3.9: The performance of CPLEX and our best version (1).

Data	#Opt (45)			Total time (s)		
	Default	**AutoBD**	**Papa**	**Default**	**AutoBD**	**Papa**
30_1	5	5	5	15	9	7.6
30_2	5	4	5	120	290.4	261
30_3	0	3	1	900	624	777.8
40_1	5	5	5	30.8	48.6	102
40_2	5	4	2	291.2	677.4	727.2
40_3	0	1	0	900	819	904
50_1	5	5	5	50.8	57.8	66.4
50_2	0	0	0	900	900	901.8
50_3	0	0	0	900	900	902.6
Total	25	27	23			

Table 3.10: The performance of CPLEX and our best version (2).

This information is presented in Table 3.11. Columns *%DiffLB* show the information about the relative percentage gaps of the lower bounds of *Papa* compared to those of *AutoBD* in each combination. We present the worst, the average, and the best of those relative percentage gaps from five data instances of each combination in Columns *Worst*, *Average*, and *Best*, respectively. Similarly, Columns *%DiffUB* represent the information about the upper bounds. Here, a positive value in *%DiffUB* and *%DiffLB* indicates that our algorithm has a better performance, i.e., it has a lower upper bound and a higher lower bound, respectively. Moreover, we count the total number of data instances that *Papa* performs at least as well as *AutoBD* to find the best integer solution, i.e., the data instances whose *%DiffUB* is greater than -0.01%. This information is shown in Columns *#Good (45)*, where the number in brackets shows the maximum total of *#Good*. In this case, the maximum total number is the total number of instances, as it is possible to perform as well as *AutoBD* in every data instance. Moreover, for each combination, Columns *#Better* present the number of data instances whose upper bound of *Papa* is strictly better or *%DiffUB* is greater than 0.01%. For each combination, the number of *#Better* in Column *Papa* cannot exceed the corresponding number in Column *Max* which relates to the number of instances for which *AutoBD* can find the optimal solution (shown in Columns *#Opt (45)* of Table 3.10). For example, for 30_1, *AutoBD* can find the optimal solution in every instance so the maximum number of better cases is zero, as we cannot find any better solution in that case. Finally, we sum the number of *#Good* and *#Better* from every combination to show the overall corresponding results in the last row.

Regarding the lower bound, it is obvious from Columns *%DiffLB* that *AutoBD* tends to have a higher lower bound on average, especially in the complicated combinations including 30_3, 40_3, and 50_3. For these combinations, the worst cases of our algorithm are at least 4% worse than those of *AutoBD*, while the best cases are better by less than 1%: see Columns *Worst* and *Best*. Therefore, *AutoBD* manages to reach much better lower bounds than *Papa* during the tree search, resulting in overall smaller relative percentage gaps.

The most interesting observation here is the information related to the upper bound as it shows the real comparison of the best integer solution in each data instance. In Columns *%DiffUB*, we focus on the more challenging combinations. *Papa* performs marginally worse on average in 30_3 and 50_2. However, it can find better integer solutions in the most challenging combinations such as 30_3, 40_3, 50_2, and 50_3: see positive values in Column *Best*. This implies that for these combinations our method can find better integer solutions than CPLEX.

The results from Columns *#Good (45)* and *#Better* prove that our algorithm is at least comparable with CPLEX in finding the best integer solutions and tends to have better performance in more complicated data instances. Column *#Good (45)* shows that *Papa* finds the best integer solutions that are at least comparable to those of *AutoBD* in every data instance of the less difficult combinations (for example, in 30_1, 30_2, 50_1) and at least three out of five instances in the more complicated combinations. In particular, 38 out of 45 instances have at

least comparable best integer solutions to those of *AutoBD*. When data instances become more complicated, our method manages to find better integer solutions: see Columns *#Better*. More specifically, we find better integer solutions than those of *AutoBD* in 50% of the total number of possible cases.

3.6.6 Conclusion

In conclusion, we developed the branch-and-Benders' cut algorithm using various sophisticated techniques. We analysed and discussed their effectiveness. We showed how each technique improves the performance of the algorithm. Finally, we compared our best version with the Benders' decomposition algorithm implemented by CPLEX. In general, our algorithm worked well before branching the root node but struggled to reach a higher lower bound at the end of the tree search, resulting in worse average percentage gaps. However, it performed at least as well as CPLEX to find the best integer solutions and even better in the most challenging data instances. Therefore, our Benders' algorithm is highly competitive with the built-in Benders' algorithm in CPLEX.

Data	TimeRN (s)		%DiffLB			%DiffUB			#Good (45)	#Better	
	Papa	AutoBD	Worst	Average	Best	Worst	Average	Best		Papa	Max
30_1	0.30	1.22	-0.01	0.00	0.01	0.00	0.00	0.00	5	0	0
30_2	0.95	2.67	0.00	0.46	2.29	0.00	0.00	0.00	5	0	1
30_3	2.29	5.90	-4.24	-2.03	0.00	-3.84	-0.74	0.20	3	1	2
40_1	0.69	2.63	0.00	0.00	0.00	0.00	0.00	0.00	5	0	0
40_2	1.60	5.20	-0.94	-0.35	0.11	0.00	0.00	0.00	5	0	1
40_3	3.82	11.57	-5.10	-2.40	1.00	-3.17	0.55	3.54	3	3	4
50_1	1.10	3.73	0.00	0.00	0.00	0.00	0.00	0.00	5	0	0
50_2	2.55	8.59	-0.84	-0.27	0.33	-0.26	-0.05	0.09	3	1	5
50_3	7.69	18.42	-5.04	-1.44	0.99	-0.56	1.35	2.49	4	4	5
Total									38	9	18

Table 3.11: Further information on the comparison between CPLEX and our best version.

Chapter 4

Tabu Search for the Scheduling Part

This chapter introduces the basic concepts of a popular meta-heuristic method called *tabu search*, followed by all details of its implementation in our model. For more comprehensive information about the method, interested readers are referred to Gendreau & Potvin (2019).

Heuristics are methods used for generating good feasible solutions quickly or, at least, within a reasonable amount of time. Therefore, they are attractive to apply to large-scale or hard-to-solve optimisation problems. However, there is no guarantee of obtaining the optimal solution. Moreover, each heuristic method is usually specific to each problem's context. *Metaheuristics* are slightly different, as they provide a general framework to implement the procedures including a list of all necessary parameters.

Numerous metaheuristic procedures have been proposed and implemented to solve districting problems: for example, simulated annealing (Browdy 1990, D'Amico et al. 2002); genetic algorithms (Forman & Yue 2003, Bação et al. 2005, Tavares-Pereira et al. 2007, Steiner et al. 2015); and the Greedy Randomized Adaptive Search Procedure (Ríos-Mercado & Fernández 2009, Fernández et al. 2010, de Assis et al. 2014, Ríos-Mercado 2016). Tabu search is one of these approaches and shows successes in many combinatorial optimisations: see Gendreau & Potvin (2019). The method has several advantages. According to Gendreau & Potvin (2019), it has a simple concept and can deal with complicating constraints easily, so it is highly practical. Also, its main features exploit the search history to guide the search to promising areas and to avoid it being trapped in local optima. Due to these attractive characteristics of tabu search, we decide to implement it in our model.

According to Gendreau & Potvin (2019), a tabu search algorithm is based on a local search technique which can be described as the following. A current solution is slightly changed by some modifications or *moves*. A set of the modified solutions is called a *neighbourhood*. Note that a neighbourhood is a subset of *search space* which contains all possible solutions that can visit during the search. The local search investigates the best solution in the neighbourhood. If it finds an improving solution, it will accept the solution as a new current solution and then repeat the process. Otherwise, it stops. Therefore, it is usually stuck in a local optimum. Tabu search is slightly different as it accepts a non-improving solution in the search. Moreover, the main feature of tabu search is to keep track of recent solutions by creating a list of forbidden moves for the next iterations. The main reason is to forbid the algorithm to return to these recent solutions for several iterations, thereby allowing it to escape from a local optimum. The list is called *tabu list* and elements in the list are called *tabus*. Tabus are stored in the list for several iterations. Their duration to stay in the list can be random or can depend on the length of a tabu list. The length of a tabu list is usually fixed but also can vary during the search. The information to be collected as a tabu and the length of a tabu list considerably affects the efficiency of tabu search, so they should be chosen carefully. Although a tabu solution (a solution that contains tabu(s)) is generally not allowed in the neighbourhood, there is an exception; it is allowed if it can improve upon the incumbent (the currently best overall solution of the algorithm), i.e., become a new incumbent. The reason is that this tabu solution provides

a new overall best solution that has not been visited before. This step is known as the *aspiration criterion*.

Although the basic concept of tabu search seems effective, the method can still struggle to deal with challenging problems with realistic-sized instances. Therefore, additional features are required to improve the quality of the search. When the size of the neighbourhood is large, it usually affects the computational time to explore the whole neighbourhood. In this case, restricting the search to only its promising areas saves time and improves the effectiveness of the algorithm. Another common feature is using multiple neighbourhoods in the same search. Each neighbourhood usually focuses on a specific area of the search space in which to improve the current solution and might perform differently in terms of effectiveness for a specific data instance (Jin et al. 2012). In particular, some neighbourhoods might perform better than others in certain circumstances due to their specific search trajectories. Therefore, using multiple neighbourhoods can exploit the strength of each neighbourhood to guide the search more effectively. A way to manage multiple neighbourhoods during the search is, therefore, important. The multiple neighbourhoods are usually run sequentially, i.e., searching a neighbourhood one by one (Ceschia et al. 2011, Wu et al. 2016, Soto et al. 2017). Some studies randomly select a neighbourhood to be explored in each iteration where each neighbourhood has a fixed probability for being chosen (Jin et al. 2012, Xia et al. 2018). Some combine several specific neighbourhoods to enhance solutions in certain stages of the algorithm: see Nanry & Barnes (2000), Jin et al. (2012), and Alrajhi et al. (2016).

For some problems, computing the objective function(s) is very time-consuming and results in a significant computational burden when having to evaluate each candidate in a large-sized neighbourhood. In this case, a surrogate objective function, which closely relates to the real objective function but requires less computational time, can help to evaluate candidates in the neighbourhood and select the most promising ones. Then, these promising candidates are assessed by the real objective function to find the best one. These processes speed up the algorithm considerably since now the real objective function is only calculated for a small number of candidates.

More advanced features of tabu search are intensification and diversification. Intensification focuses more on promising parts of the search space. A typical way to identify the promising areas is to observe which components of the current solution have appeared in most iterations. These components can be seen as attractive to explore more thoroughly. The intensification process usually starts with fixing these attractive components in the currently best-known solution and then restores the remaining parts of the solution. This way guides the search to stay around the promising areas. However, intensification is not necessary if the algorithm with the normal process searches thoroughly enough and does not miss good solutions (Gendreau & Potvin 2019). The opposite of intensification is diversification; it prevents the search from focusing only on some specific parts of the search space and guides the algorithm to seek better solutions in other areas. Therefore, it forces the search to explore yet unvisited areas. A typical example of this is to allow the algorithm to accept infeasible solutions during the search. Most studies often deal with infeasible solutions by introducing self-adjusting penalties that rely on search history. Kulturel-Konak et al. (2004) confirmed that adaptive penalties from the search history are more robust than constant ones. Another typical method is destroying some parts of the current solution and then restoring it from scratch. This feature is often applied after the search cannot improve the current solution for a certain amount of time or the number of iterations. Gendreau & Potvin (2019) suggested that diversification is crucial to a successful algorithm, as if it is not appropriately implemented, it can easily fail to diversify solutions adequately.

For districting problems, Bozkaya et al. (2003) proposed tabu search to solve a political districting problem with a multi-criteria objective function. Here, a basic unit and a district represent a census tract and an electoral constituency, respectively. The move operations in this work move one basic unit to its adjacent district and swap two basic units between adjacent districts. The algorithm allows accepting infeasible solutions that violate population equality, which is one of the criteria, and penalises them with a self-adjusting penalty multiplier based on the search history. They diversified solutions by keeping track of frequent non-improving moves and penalising them. This is to discourage deteriorating moves in the search. They also applied an additional advanced feature called adaptive memory. It was originally proposed by

Rochat & Taillard (1995) to create a good solution by exploiting components of the best-known solutions. Bozkaya et al. (2003) applied the developed method to real-life data of Edmonton, Canada and proved that it could find a better districting plan than existing ones.

The algorithm by Bozkaya et al. (2003) was also applied successfully in the following papers. Blais et al. (2003) adapted the algorithm to implement a real design for health clinic districts in Montreal, Canada, where the ease of travel by public transportation is one of their criteria. It showed highly satisfactory results after two years of the plan's management. Wei & Chai (2004) combined the tabu search algorithm with a scatter search and showed success in improving the quality of solutions for a multi-objective spatial zoning problem. Haugland et al. (2007) created districts for a vehicle routing problem, where customers' demands are uncertain during the design of the districts. The authors showed that tabu search outperforms a multi-start heuristic. Bozkaya et al. (2011) integrated the algorithm with the Geographic Information System to revise a political districting plan in Edmonton, Canada. The proposed plan was officially approved for a real election in 2010.

Ríos-Mercado et al. (2017) proposed a tabu search algorithm to design districts of recollecting waste of electrical and electronic equipment for companies who are responsible for recycling. The authors formulated the problem as a maximum dispersion territory design problem to prevent regional monopolies. Infeasible solutions are allowed in the search and controlled by self-adjusting penalties. The empirical results confirmed that the tabu search algorithm is superior to the Greedy Randomised Adaptive Search Procedure from Fernández et al. (2013), which is a recent state-of-the-art algorithm for the problem. Another recent work is by Gliesch et al. (2018). They extended a study from Rios-Mercado & Escalante (2016) that creates districts for commercial products distribution. Note that the compactness in this work is not measured from district centres, but instead from the maximum Euclidean distance between two nodes in a district. They implemented tabu search to improve each criterion separately. In particular, they run tabu search for maximising compactness first and then reducing imbalance in the districts. The results show the advantages of implementing tabu search for improving a specific criterion instead of using a single search with a multi-criteria objective function.

The most interesting work for us in this area is Butsch et al. (2014). The authors proposed a tabu search algorithm to design districts for arc routing with a multi-criteria objective function. Here, a basic unit is represented by an edge of the graph. This problem can be seen in the application of mail or leaflet delivery. They used the same move operations as in Bozkaya et al. (2003), i.e., shifting a basic unit between its adjacent districts. The algorithm applies multiple neighbourhoods where each neighbourhood is based on a strategy to improve a specific criterion. In each iteration, a neighbourhood is selected randomly corresponding to its self-adjusting probability. The probabilities of neighbourhoods depend on the performance of solutions from these neighbourhoods in recent iterations. If any criterion needs improvement, the probability of selecting the neighbourhood for the corresponding criterion will be increased. This is to raise the chance of searching the neighbourhood on solutions in the next iterations, thereby improving the corresponding criterion. We believe that this is the first work that adjusts a probability to select each neighbourhood based on the search history, while most studies with a random selection on multiple neighbourhoods in the literature choose a neighbourhood with a fixed probability. They also presented an effective way to improve compactness, which is one of their criteria. They selected districts with the highest compactness. Then, in each of these districts, they removed basic units with the largest distances from the district centre. In the case of swapping basic units, the authors avoided deteriorating solutions by removing a basic unit and then inserting a basic unit with a smaller distance from the district centre. They tested on real street data and showed the effectiveness of the algorithm.

Tabu search has also been successful in tackling closely related applications to district problems, like facility location, network design, and especially vehicle routing problems. Michel & Van Hentenryck (2004) proposed tabu search to solve an uncapacitated facility location problem. They presented a simple move to create a neighbourhood: changing a status (closed or open) of a warehouse in each iteration. The tabu list in their work recorded a list of warehouses that have changed their status. The idea for diversification is also simple, with randomly closing open warehouses. Despite its simplicity, the algorithm showed competitive results compared to those for state-of-the-art algorithms. With a similar kind of moves and tabu list, Gendron et al. (2003) solved a multicommodity capacitated location problem. The problem requires the

simultaneous decision on opening depots and the corresponding network flow for a fleet of containers. There are two neighbourhoods in this case; one for changing the status of a depot (close or open) and another one for swapping different states of two depots. The authors searched the former neighbourhood and then the latter alternatingly. For each move, the corresponding minimum cost network flow problem is then solved. Since the computational time for exploring the whole neighbourhoods is excessive, the authors approximated the change in the cost of each move and then solved the minimum cost network flow problem only for promising cases. Cortinhal & Captivo (2003), who combined tabu search with a Lagrangian heuristic to improve the upper bound of a capacitated facility location problem, presented different move operators. The moves focus on the allocation of customers to open plants, i.e., shifting a customer to another open plant, or swapping the allocation of two customers who have been allocated to different plants. The tabu list, in this case, records the customer and their previously allocated plant. The authors also compared the performance when the tabu search is substituted with a local search and showed that the tabu search is better. More recent work in this area is by Yaghini et al. (2013) who proposed tabu search combined with the cutting plane method to solve a capacitated p-median problem. The move operator here is closing a currently open facility. Then, the corresponding LP relaxation of the capacitated p-median is solved and strengthened by the cutting plane method. The results showed that this method outperformed the best-known algorithm in the literature of the problem.

According to Gendreau & Potvin (2019), tabu search is one of the most effective metaheuristics to solve capacitated vehicle routing problems. Typical move operators are moving a node (which is usually represented by a city or a customer to visit) within the same route and between routes. Gendreau et al. (1994) allowed infeasible solutions during the search and managed them by self-adjusting penalties. The authors proposed a robust way to insert a node into a new route: moving a node to a route that contains one of its closest neighbouring nodes. They tested the algorithm on instances consisting of between 50 and 200 nodes and showed that this way produces high-quality neighbouring solutions. Paquette et al. (2013) focused on a multi-criteria dial-a-ride problem which aims for a high-quality design of vehicle routes for picking up and delivering passengers. It is considered as an extension of capacitated vehicle routing problems. The objective function for this problem is a linear combination of several weighted functions where each of them is represented for each criterion. The weights for the criteria in the objective function are updated dynamically during the search based on the search history. Any criterion on the current solution that needs to be improved gets a higher weight, and this enforces the search to emphasise this specific criterion. The authors presented the results on real-life data of Montreal to show the effectiveness of the algorithm. Jin et al. (2012) showed the advantages of using multiple neighbourhoods with multiple threads. Each thread can be either a single neighbourhood or a certain combination of multiple neighbourhoods to improve a solution in a specific way. In a thread that has multiple neighbourhoods, the authors assigned a certain probability of choosing each neighbourhood. Xia et al. (2018) also used tabu search with multiple neighbourhoods to solve a capacitated vehicle routing problem that allows splitting customers' orders to different vehicles. They selected a neighbourhood randomly in each iteration where each neighbourhood has the same probability of being chosen.

Next, we propose a general framework of the tabu search algorithm for our model with additional features to improve the quality of the search in Section 4.1. In particular, we allow infeasible solutions during the search to explore more areas in the search space. We propose three neighbourhoods for our problem as presented in Section 4.2−4.3. Since the size of neighbourhoods is usually large, we explore only specific promising areas of the neighbourhoods and use a surrogate objective function when the computational cost of evaluating the actual objective function is too expensive. Also, we use a simple but effective diversification scheme. After introducing all neighbourhoods, Section 4.4 presents a way to use multiple neighbourhoods in the same search. In particular, we randomly select a neighbourhood in each iteration corresponding to self-adaptive probabilities. The details of setting up the experiment are presented in Section 4.5. The effectiveness of the proposed algorithm is discussed through extensive computational results in Section 4.6.

4.1 General Framework of Tabu Search for our Model

To formulate the algorithm, we represent a solution by the information based on customers who are week centres and a week pattern of each customer, since these are enough to define a district each week. We create a vector of customers who are week centres each week, called ω, and a vector of week patterns of customers, called ρ, in the following ways:

- $\omega \equiv (\omega_1, \omega_2, \ldots, \omega_{|W|})$, where ω_w is the customer who is the week centre in week w for $w = 1, 2, \ldots, |W|$.
- $\rho \equiv (\rho_1, \rho_2, \ldots, \rho_{|B|})$, where ρ_b is the week pattern of customer $b \in B$.

We define $S \equiv (\omega, \rho)$ as a solution formed by a vector of week centres and week patterns. We also increase the search space by allowing infeasible solutions in the search, i.e., we accept a solution that violates the workload balance. A degree of the violation for a vector of week patterns ρ, $\delta(\rho)$, is measured by the total excess of the workload from the acceptable ranges during the planning horizon.

It is straightforward to derive the vector ω and ρ from the week centre and week pattern variable values, respectively:

- $\omega_w = b$, where $x_b^w = 1$, $b \in B$, $w \in W$.
- $\rho_b = p$ such that $g_{bp} = 1$, $b \in B$, $p \in P_b$.

$\delta(\rho)$ is also derived easily. Let $\delta^w(\rho)$ represent the excess of the workload in week $w \in W$, i.e.,

$$\delta^w(\rho) = \max\left\{0,\ (1-\tau^{week})\mu^{week} - \sum_{b\in B} t_b \psi^w_{\rho_b},\ \sum_{b\in B} t_b \psi^w_{\rho_b} - (1+\tau^{week})\mu^{week}\right\}.$$

Then, $\delta(\rho) = \sum_{w\in W} \delta^w(\rho)$ and if it is positive, it indicates that the solution is infeasible due to a violation of the acceptable workload ranges.

We focus on applying tabu search to the model by creating neighbourhoods based on either the week centre or the week pattern vector. In the week centre neighbourhood, we modify the week centre vector and then derive the corresponding week pattern vector to form a neighbouring solution. Similarly, a neighbouring solution in the week pattern neighbourhood is created from a modified week pattern vector and the corresponding week centres. In other words, while exploring the neighbourhoods, we either have to find the optimal vector of week patterns for a given week centre vector or vice versa. The model formulation of each problem is almost the same as the allocation problem and the location problem to find an initial integer solution for Benders' decomposition in the previous chapter. Note that the location problem is to find the best week centres of the planning horizon for given week patterns of customers, while the allocation problem is the other way round. The only differences are that each problem is represented by either the predefined week centre or the week pattern vector, and has the penalisation of the workload violation in the objective function.

Regarding the former problem for a given week centre vector $\overline{\omega}$, we solve the following problem (Allocation($\overline{\omega}$)) to get the optimal week patterns ρ. To penalise an infeasible solution, we introduce new continuous variables for the allocation problem δ_U^w and δ_L^w as the excess workload from the upper bound and the lower bound of the acceptable range in a week $w \in W$, respectively. Then, the penalty of the solution is the total excess of workload weighted by a penalty factor ϕ. In this case, $\delta(\rho)$ can be simply derived from the optimal value of

$\sum_{w\in W}(\delta_U^w + \delta_L^w)$.

$$\begin{aligned}
\min \quad & z_{AL}(\overline{\omega}) := \sum_{b\in B}\sum_{w\in W}\sum_{p\in P_b} c_{b,\overline{\omega}_w}\psi_p^w g_{bp} + \phi\sum_{w\in W}(\delta_U^w + \delta_L^w) && \text{(Allocation}(\overline{\omega}))\\
\text{s.t.} \quad & \sum_{p\in P_b} g_{bp} = 1 && b\in B\\
& \sum_{b\in B, p\in P_b} t_b\psi_p^w g_{bp} \geq (1-\tau^{week})\mu^{week} - \delta_L^w && w\in W\\
& \sum_{b\in B, p\in P_b} t_b\psi_p^w g_{bp} \leq (1+\tau^{week})\mu^{week} + \delta_U^w && w\in W\\
& g_{bp}\in\{0,1\} && b\in B,\ p\in P_b\\
& \delta_U^w, \delta_L^w \geq 0 && w\in W.
\end{aligned}$$

The other problem of a fixed vector of week patterns follows the same idea. Given a vector of week patterns $\overline{\rho}$, we solve the following problem (Location($\overline{\rho}$)) to derive the second component, the vector of week centres, for a solution. Also, the penalty value is added to the objective value in case of infeasible week patterns.

$$\begin{aligned}
\min \quad & z_L(\overline{\rho}) := \sum_{b\in B}\sum_{i\in B}\sum_{w\in W} c_{bi}u_{bi}^w + \phi\overline{\delta}(\overline{\rho}) && \text{(Location}(\overline{\rho}))\\
\text{s.t.} \quad & \sum_{i\in B} u_{bi}^w = \psi_{\overline{\rho}_b}^w && b\in B,\ w\in W\\
& \sum_{b\in B} x_b^w = 1 && w\in W\\
& u_{bi}^w \leq x_i^w && b,\ i\in B,\ w\in W\\
& u_{bi}^w \geq 0 && b,\ i\in B,\ w\in W\\
& x_b^w \in\{0,1\} && b\in B,\ w\in W.
\end{aligned}$$

This problem is solved to a similar way of the location phase for an initial integer solution in Benders' decomposition. Let $B_w = \{b\in B \mid \psi_{\overline{\rho}_b}^w = 1\}$ be the set of customers in week $w\in W$ corresponding to $\overline{\rho}$. The week centre ω can be derived by solving the following:

$$\omega_w = \arg\min_{i\in B}\sum_{b\in B_w} c_{bi} \qquad w\in W \tag{4.1}$$

and the objective value of the location problem is

$$z_L(\overline{\rho}) = \sum_{w\in W}\sum_{b\in B_w} c_{b,\omega_w} + \phi\overline{\delta}(\overline{\rho}). \tag{4.2}$$

A general framework for a tabu search technique for week centre and week pattern neighbourhoods is described as follows. Given a current solution, the algorithm creates a neighbourhood without tabu solutions from the current solution, explores the neighbourhood to find the best neighbouring solution, and records the information of the best neighbouring solution in a tabu list. Since the tabu list is fixed for this algorithm, if the list is full we will add a new tabu and remove the oldest tabu from the list. As we described above, we have a penalty factor to penalise an infeasible solution. Whenever we get consecutive infeasible solutions for the predefined number of iterations, the value of the penalty factor increases. The factor is multiplied by a positive constant value to discourage this situation to happen again.

Moreover, a diversification scheme is applied if one of the following situations happen: the current neighbourhood is empty, the penalty factor reaches the maximum fixed value, or there is no improvement of the value of an incumbent for a fixed number of consecutive iterations. In the first case, it is clear that we do not have any solution to move to in the next iteration, so we diversify the current solution and update it as the current solution in the next iteration. In the second case, the increasing value of the penalty factor indicates that the solution remains

infeasible, while the last case shows that the solution might be struggling to escape a local optimum. For such cases, the best neighbouring solution is diversified and the tabu list is cleared. We hope that diversification helps the algorithm to overcome all these situations. At the end of each iteration, the best neighbouring solution is the new current solution for the next iteration and the algorithm repeats, until it reaches a stopping criterion. The stopping criteria are the maximum total time and the maximum number of consecutive diversifications without a new incumbent.

The core algorithm is shown in Algorithm 9. An initial solution S^0 is required to start the procedures as it will be the first current solution. All parameters to control the algorithm are as follows. Regarding infeasible cases, $k > 1$ is a constant value to strengthen the penalty factor whenever the total number of consecutive infeasible solutions reaches N_V. The maximum value of the penalty factor, ϕ^*, and the maximum iterations for obtaining consecutive non-improving solutions, N_{NI}, are the parameters for the criteria of the diversification. N_D and $maxTime$ represent the maximum number of consecutive diversifications and maximum total time for the stopping criteria, respectively.

As the algorithm progresses, we create a neighbourhood from a current solution $\overline{S}$ and search for its best neighbouring solution S'. i, v, and j are used to count the total number of successive iterations to get a non-improving solution, an infeasible solution, and a diversification, respectively. ϕ is the current penalty factor, whose initial value is 1. $Time$ represents the current time of the algorithm and is updated at the end of each iteration. When the algorithm reaches any stopping criteria, it returns the incumbent solution S^*.

To obtain an initial solution for the tabu search algorithm, we apply the location-allocation heuristic similar to that for the Benders' decomposition: select a week centre in each week randomly among customers whose week rhythm is the smallest in the planning horizon, and then solve the allocation and the location problem iteratively. Here, we allow an infeasible solution in the location and the allocation problems, and if the tabu search algorithm starts from an infeasible solution, it usually struggles to find a feasible solution. Therefore, we increase the value of the penalty factor to be significantly larger in the location-allocation heuristic to discourage the initial solution from becoming infeasible. On rare occasions, the initial solution might still be infeasible after solving the location-allocation heuristic. For that case, we repeat all of the processes with a new initial week centre vector which is not entirely identical to the previous one. We allow the repetition for at most a fixed number of times. After we derive an initial solution (which should be feasible), we change the penalty factor to its initial value and begin the tabu search.

For the diversification, we randomly change some parts of a solution and then apply the location-allocation heuristic to improve its quality. Again, we repeat the process of the diversification at most a fixed number of times until we get a feasible solution.

To evaluate the performance of the tabu search algorithm, for each data instance, we compare the relative percentage gap to the best-found solution from the previous chapter, i.e., the best solution from either the modern brand-and-cut in CPLEX, the Benders' decomposition algorithm in CPLEX, or our developed Benders' decomposition algorithm. Let z_{BF} and z_H be the objective function values from the best-found solution and the heuristic, respectively. The relative percentage gap $\%Gap$ is shown below:

$$\%Gap = \frac{(z_{BF} - z_H) \times 100}{z_{BF}}. \tag{4.3}$$

Basically, a nonnegative value of the measurement suggests that the heuristic performs at least as good as the benchmark, or even better, while a negative value identifies the worse performance of the heuristic.

In the case when CPLEX can find an optimal solution, we aim to get the gap nearly to zero since this indicates that a solution of the heuristic is as good as the optimal one. For data instances in that CPLEX cannot find an optimal solution, we ideally want to obtain a positive gap which means that the heuristic has found a better solution with a smaller objective value.

In the following sections, we provide details of the neighbourhoods on week centres and week patterns. Moreover, for each neighbourhood, we propose a method to diversify a current solution when the algorithm struggles to find a new incumbent or escape infeasibility.

Algorithm 9 The tabu search algorithm

Input: S^0
Parameter: k, N_V, ϕ^*, N_{NI}, N_D, $maxTime$
Initial:
$\overline{S}$:= S^0, S^*:= S^0, $i, j, v := 0$, $\phi = 1$, $Time := 0$
while $Time < maxTime$ **do**
  Find the best non-tabu solution S' in the neighbourhood of $\overline{S}$
  if S' is not empty **then**
    Update the tabu list with the information of S'
    if S' is infeasible **then**
      $v := v + 1$ ▷ Increase the counter of consecutive infeasible cases
      $i := i + 1$ ▷ Increase the counter of consecutive non-improving cases
      if $v \geq N_V$ **then**
        $\phi := k \cdot \phi$ ▷ Strengthen the value of the penalty factor
        $v := 0$ ▷ Clear the counter for infeasible cases
      end if
    else
      $v := 0$, $\phi = 1$ ▷ Clear all parameters related to the infeasible cases
      if S' is better than S^* **then**
        $S^* := S'$
        $i := 0$, $j := 0$ ▷ Clear the counters for non-improving cases
      else
        $i := i + 1$ ▷ Increase the number of consecutive non-improving cases
      end if
    end if
  end if
  if S' is empty **or** $i \geq N_{NI}$ **or** $\phi \geq \phi^*$ **then**
    $j := j + 1$ ▷ Increase the number of consecutive diversifications
    if $j > N_D$ **then** ▷ Check the stopping criterion
      Stop the algorithm
    else
      Clear the tabu list
      Apply the diversification and derive a new solution for S'
      $v := 0$ ▷ Reset the total number of the infeasible cases
      $i := 0$ ▷ Clear the counter of the non-improving cases
      if S' is better than S^* **then**
        $S^* := S'$ ▷ Update the incumbent if the diversification can find a better one
      end if
    end if
  end if
  $\overline{S} := S'$
  Update the current time in $Time$
end while
Output: S^*

4.2 Week Centre Neighbourhood

Given a current solution $S \equiv (\omega, \rho)$, we create a neighbourhood on the week centre vector ω as follows. Starting from the first week centre ω_1, we change it to another customer $b \in B \setminus \{\omega_1\}$, while keeping all other week centres fixed, resulting in a modified week centre vector $\overline{\omega}$. Then, we solve (Allocation($\overline{\omega}$)) to determine the optimal week patterns, the corresponding total excess of workload, and the total centre-based distances. We repeat the procedures with every customer in $b \in B \setminus \{\omega_w\}$ and every week $w = 1, \ldots, |W|$. In the end, the best neighbouring solution is stored. We call this local search the *exhaustive search on week centres* (EWC).

For the best solution in the neighbourhood, the tabu list records the week that has a new week centre. In the next iteration, we do not change the week centres of any week in the tabu list, to avoid cycling back to recent solutions from previous iterations. We define the length of the tabu list as $l = \left\lfloor \frac{|W|}{2} \right\rfloor$, i.e., equal to half of the length of the planning horizon. Therefore, we freeze each week in the tabu list for at most l iterations, before removing it from the list.

However, the algorithm might still struggle to escape from a local optimum. For such a case, we apply a diversification step in the algorithm: l week centres are randomly changed to other customers. Then, we run the location-allocation algorithm to derive a final diversified solution. Note that we repeat the whole process at most a fixed number of times until we get a feasible solution.

Algorithm 10 shows the steps for the EWC algorithm with the corresponding tabu list T_{WC}. For each week, every customer except the current week centre can be a candidate for a new week centre, as shown in Line 2.

Algorithm 10 Exhaustive Search in the Week Centre Neighbourhood (EWC)

Input: $S \equiv (\omega, \rho)$, T_{WC}
Initial:
    $z^* := 1e10$
    $S' \equiv (\omega', \rho') := S$
1: **for all** $w \in W$ such that $w \notin T_{WC}$ **do**
2: **for all** $b \in B \setminus \{\omega_w\}$ **do**
3: $\omega' := \omega$
4: $\omega'_w := b$ ▷ Set the alternative week centre
5: Solve (Allocation(ω')) and derive $z_{AL}(\omega')$ and ρ'
6: **if** $z_{AL}(\omega') < z^*$ **then**
7: $L^* := w$ ▷ Update the tabu
8: $z^* := z_{AL}(\omega')$
9: $S^* := S'$ ▷ Update the best solution that has been found so far
10: **end if**
11: **end for**
12: **end for**
13: Add element L^* to T_{WC} (if the neighbourhood is not empty) ▷ Update the tabu list
    Output: S^*

4.2.1 Improving the Search in the Week Centre Neighbourhood

Reduced-size Neighbourhood

Since the size of the neighbourhood is usually large for even small data instances, we aim to reduce its size with a guarantee of good solutions. Our preliminary results show that the new week centre for the best neighbouring solution is often either one of the other week centres, or at least geographically close to them. The way to find close customers will be provided later in the section. We can restrict our search to any customers who fall into one of the two categories.

To create the reduced-size neighbourhood, we limit the number of customers to investigate each week and guarantee that they belong to one of the above two categories. Assume that the current solution is $S \equiv (\omega, \rho)$. For week $w \in W$ and customer $i \in B \setminus \{\omega_w\}$, we calculate the

minimum distance d_i^w from i to all of the other week centres, i.e.,

$$d_i^w = \min_{j \in W \setminus \{w\}} c_{i,\omega_j}. \tag{4.4}$$

Then, we create the list L^w for week $w \in W$ by sorting all customers in non-decreasing order of d_i^w. As a result, other week centres and any customers close to them make up the first elements. To reduce the size of the neighbourhood, let $\alpha \in \mathbb{R}$, $0 < \alpha < 1$, be the parameter for the reduction, and $N_\alpha = \lfloor \alpha \cdot |B| \rfloor$ be the reduced number of customers to investigate each week. Different values of α will be tested in the numerical experiments. We store the first N_α customers of L^w in the set B_α^w. We call this the *α-reduced-size search on the week centre neighbourhood* (α-WC). In this case, everything in Algorithm 10 remains unchanged except for Line 2, where we restrict the search for a new week centre in week w to the set of customers in B_α^w.

Quick Allocation Problem (QAP)

However, in the preliminary results, even with the reduced-size week centre neighbourhood, it can take a reasonably long time in each iteration. The reason for this is that we still have to solve the allocation problem for every week centre candidate, which is a costly operation. Therefore, we propose another idea to determine a surrogate objective value for the allocation problem instead. Namely, we omit solving the allocation problem for the new week pattern vector, and just compute the total centre-base distances with the week pattern vector from the current solution. More precisely, given a modified week centre vector $\overline{\omega}$ and the current week pattern vector ρ, the approximate objective value of the allocation problem $z_{AL}^+(\overline{\omega}, \rho)$ is given as

$$z_{AL}^+(\overline{\omega}, \rho) := \sum_{b \in B} \sum_{w \in W} \psi_{\rho_b}^w c_{b,\overline{\omega}_w} + \phi\delta(\rho). \tag{Allocation$^+(\overline{\omega}, \rho)$}$$

We select the best modified week centre vector based on the approximate values and then solve the allocation problem for this vector to get the corresponding optimal week patterns. This reduces the computational time significantly and becomes quick already even for the exhaustive search, because the integer programme has to be solved only once during the search of the neighbourhood. We call this technique the *Quick Allocation Problem (QAP)*.

The exhaustive search with QAP is presented in Algorithm 11. There are a few differences from the previous algorithm. Line 5 shows that the approximate objective value of the allocation problem is used for finding the best modified week centre vector in the search. Then, the allocation problem is solved just once for the best week centre vector: see Line 13.

4.3 Week Pattern Neighbourhoods

To create a neighbourhood by changing the week patterns of customers, it is not necessary to consider customers who require weekly services. So, we construct a neighbourhood only among customers whose week rhythm is at least two. There are two neighbourhoods on a vector of week patterns; either by *switching week patterns* or *swapping week patterns*. The former idea chooses one customer at a time and changes their pattern to another one. The latter considers two customers who have the same week rhythm, but different week patterns, and then swaps their week patterns. We follow the same idea as in the week centre neighbourhood: after getting a modified week pattern vector $\overline{\rho}$, we solve the location problem to get the optimal week centres and the optimal objective function value. As we mentioned before, we can derive the optimal week centre of each week manually from (4.1), without having to solve (Location($\overline{\rho}$)).

After finding the best neighbouring solution, we store the customer whose week pattern changes and their original week pattern in a tabu. This is to guarantee that in the following iterations the algorithm does not consider the customer with their previous week pattern. For the neighbourhood of switching week patterns, a tabu is given by (b, p), where b is the customer whose current week pattern p is changed. In the case of swapping week patterns, tabus are the original week patterns of the two customers before swapping their patterns. That is, for

Algorithm 11 Exhaustive Search with QAP in the Week Centre Neighbourhood (EWC-QAP)

Input: $S \equiv (\omega, \rho)$, T_{WC}
Initial:
$z^* := 1e10$
$S' \equiv (\omega', \rho') := S$
1: **for all** $w \in W$ such that $w \notin T_{WC}$ **do**
2: **for all** $b \in B \setminus \{\omega_w\}$ **do**
3: $\omega' := \omega$
4: $\omega'_w := b$ ▷ Update a new week centre
5: Calculate $z^+_{AL}(\omega', \rho)$ ▷ Calculate the approximate objective value of the allocation problem
6: **if** $z^+_{AL}(\omega', \rho) < z^*$ **then**
7: $L^* := w$ ▷ Update the tabu
8: $z^* := z^+_{AL}(\omega', \rho)$
9: $\omega^* := \omega'$ ▷ Update the best new week centre vector by this criterion
10: **end if**
11: **end for**
12: **end for**
13: Solve (Allocation(ω^*)) and derive $z_{AL}(\omega^*)$ and ρ^* for S^* ▷ Update the corresponding week patterns
14: Add element L^* to T_{WC} (if the neighbourhood is not empty) ▷ Update the tabu list
Output: S^*

two customers b and b' who swap their week patterns p and p', respectively, we add (b, p) and (b', p') to the tabu list.

Regarding the length of the tabu list for each neighbourhood, let $B^{NW} = \{b \in B \mid r_b > 1\}$ denote the set of customers who do not require weekly service. We initially defined the length of the tabu list for both week pattern neighbourhoods to be $\lfloor \frac{1}{2} \sum_{b \in B^{NW}} r_b \rfloor$. However, from the preliminary experiments, we found that this is too long for the tabu list of the swapping week patterns neighbourhood, as it often results in an empty neighbourhood, once the tabu list has reached its maximum size. Therefore, we decide to reduce the length of the tabu list of the swapping policy to $\lfloor \frac{1}{4} \sum_{b \in B^{NW}} r_b \rfloor$.

We propose a diversification scheme for these neighbourhoods in case that the search cannot find a better incumbent for a fixed number of consecutive iterations. We apply the same idea as for the week centre neighbourhood: $\left\lfloor \frac{|B^{NW}|}{2} \right\rfloor$ customers in the set B^{NW} are randomly chosen to change their week pattern to another, again randomly chosen one. Then, the location-allocation algorithm is run to improve the quality of the solution before we continue the tabu search algorithm. The whole process is repeated at most a fixed number of times unless we achieve a feasible solution.

The search in the switching week pattern neighbourhood is outlined in Algorithm 12, with its tabu list, T_{Switch}. The conditions for choosing candidates for a new week pattern are shown in Line 2: a new week pattern p of a customer b cannot be the same as the original week pattern ρ_b, and (b, p) does not belong to the tabu list. A new week pattern vector is given by switching the customer's week pattern to another one; see Line 4. Then, the rest of the algorithm follows the steps that we described at the beginning.

Algorithm 13 shows the search in the swapping week pattern neighbourhood, with the corresponding tabu list T_{Swap}. According to our previous discussion, the tabu list has a different length to that of the switching week pattern.

To facilitate the processes in the algorithm, we define additional notations in the following. We denote $R^{NW} = \{r_b \mid b \in B^{NW}\}$ as the set of all possible customer week rhythms, except the weekly ones. Also, let $B^r = \{b \in B \mid r_b = r\}$ be the set of customers who have week rhythm $r \in R^{NW}$. For each customer $b \in B^{NW}$, we are interested to swap their pattern with a different one. Therefore, we create a specific set for each customer b: $B^r_b = \{\overline{b} \in B^r \setminus \{b\} \mid \rho_{\overline{b}} \neq \rho_b\}$ which contains all customers who have the same frequency as b but a different week pattern. If the set is not empty, there is a possible swap for the week pattern of customer b. Moreover,

after swapping their week patterns, neither can belong to the tabu list. These restrictions are in Line 2. The rest of the algorithm is analogous to Algorithm 12.

Algorithm 12 Exhaustive Search in the Switching Week Pattern Neighbourhood (Switch)

Input: $S \equiv (\omega, \rho)$, T_{Switch}
Initial:
z^*:= 1e10
$S' \equiv (\omega', \rho')$:= S
1: **for all** $b \in B^{NW}$ **do**
2: **for all** $p \in P_b \setminus \{\rho_b\}$ & $(b, p) \notin T_{Switch}$ **do**
3: $\rho' := \rho$
4: $\rho'_b := p$ ▷ Update the week pattern of customer b
5: Solve (Location(ρ')) and derive ω' and $z_L(\rho')$ from (4.1) and (4.2)
6: **if** $z_L(\rho') < z^*$ **then**
7: $L^* := (b, \rho_b)$ ▷ Update a tabu
8: $z^* := z_L(\rho')$, $S^* := S'$
9: **end if**
10: **end for**
11: **end for**
12: Add element L^* to T_{Switch} (if the neighbourhood is not empty) ▷ Update the tabu list
Output: S^*

Algorithm 13 Exhaustive Search in the Swapping Week Pattern Neighbourhood (Swap)

Input: $S \equiv (\omega, \rho)$, T_{Swap}
Initial:
z^*:= 1e10
$S' \equiv (\omega', \rho')$:= S
1: **for all** $b_1 \in B^{NW}$ **do**
2: **for all** $b_2 \in B_{b_1}^{r_{b_1}}$ & $((b_1, \rho_{b_2})$ & $(b_2, \rho_{b_1}) \notin T_{Swap})$ **do**
3: $\rho' := \rho$
4: $\rho'_{b_1} := \rho_{b_2}$ and $\rho'_{b_2} := \rho_{b_1}$ ▷ Swap week patterns of these two customers
5: Solve (Location(ρ')) and derive ω' and $z_L(\rho')$ from (4.1) and (4.2)
6: **if** $z_L(\rho') < z^*$ **then**
7: $L_1^* := (b_1, \rho_{b_1})$ and $L_2^* := (b_2, \rho_{b_2})$ ▷ Update tabus
8: $z^* := z'$, $S^* := S'$ ▷ Update the best solution that has been found so far
9: **end if**
10: **end for**
11: **end for**
12: Add elements L_1^* and L_2^* to T_{Swap} (if the neighbourhood is not empty) ▷ Update the tabu list
Output: S^*∗

4.3.1 Restricted Search in the Week Pattern Neighbourhoods

Restricting the search of the neighbourhoods to promising areas that contain good solutions is beneficial to save computational time and enhance the efficiency of the algorithm. Therefore, we propose a simple criterion to limit the search in each week pattern neighbourhood: customers who are among the furthest from the week centres in their respective week clusters should preferably change their week pattern.

Assume now that the current solution is $S \equiv (\omega, \rho)$. To create the list of customers who should change their week pattern, we calculate for each customer $b \in B^{NW}$ the maximum distance $\hat{d}_b$ to the week centres of their current week clusters as follows:

$$\hat{d}_b = \max_{w \in W} \psi^w_{\rho_b} c_{b,\omega_w}.$$

Feature	WC	Switching WP	Swapping WP
Tabu	Week that has a new week centre.	(customer, his original week pattern)	
Tabu length	$\left\lfloor \frac{\lvert W \rvert}{2} \right\rfloor$.	$\left\lfloor \frac{1}{2} \sum_{b \in B^{NW}} r_b \right\rfloor$.	$\left\lfloor \frac{1}{4} \sum_{b \in B^{NW}} r_b \right\rfloor$.
Diversification	Change $\left\lfloor \frac{\lvert W \rvert}{2} \right\rfloor$ week centres randomly.	Change week patterns of $\left\lfloor \frac{\lvert B^{NW} \rvert}{2} \right\rfloor$ customers randomly.	
Improvement	1. Reduced search: focus only on other week centres or customers close to them. 2. QAP (Quick Allocation Problem).	Reduced search: focus only on customers who are furthest from their week centres.	
Stopping criterion	1. Maximum time. 2. Maximum number of consecutive diversifications without a new incumbent.		

Table 4.1: Summation of features of every single neighbourhood.

The list L^C contains the customers $b \in B^{NW}$ in non-increasing order with respect to $\hat{d}_b$. We focus on the first customers of the list since they are the furthest from their week centres, and are most likely to yield an improvement of the total centre-based distance. This reduced-size idea is similar to that for the week centre neighbourhood. Let $\beta \in \mathbb{R}$, $0 < \beta < 1$, be the parameter that defines the reduced proportion of all customers to change their week patterns, and $N_\beta = \lfloor \beta \cdot |B^{NW}| \rfloor$ be the maximum number of those customers. We define the list L^C_β to store the first N_β customers of the list L^C, who have a non-empty set of week patterns to change.

We call the reduced-size switching or swapping week pattern neighbourhood with the positive parameter β as the *β-reduced-size search on the week pattern neighbourhood* (β-j) where j is either Algorithm 12 or Algorithm 13 (Switch or Swap, respectively). This idea can save a lot of computational time, especially for the swapping week pattern neighbourhood whose size increases exponentially with the total number of customers. To adapt these accelerations to the original search of the week pattern neighbourhoods, we update the set of customers to change their week patterns to L^C_β in Line 1 of Algorithm 12 and Line 1 of Algorithm 13 for the switching and swapping week pattern neighbourhood, respectively. The rest of the algorithm is analogous to the exhaustive versions.

To sum up, Table 4.1 presents important features of every single neighbourhood, where *WC*, *Switching WP*, and *Swapping WP* stand for the week centre neighbourhood, the switching week pattern neighbourhood, and the swapping week pattern neighbourhood, respectively.

4.4 Mixed Neighbourhood

In this section, we propose an idea that unifies all of the above neighbourhoods. The method is inspired by the adaptive large neighbourhood search introduced by Ropke & Pisinger (2006*a*). Ropke & Pisinger (2006*a*) created neighbouring solutions by randomly choosing a heuristic among several ones in each iteration. Initially, each heuristic is assigned an equal weight. The weight of a heuristic is adjusted by its performance during the algorithm. For example, if the heuristic can improve the quality of a current solution, its weight will be increased as a reward. These weights are used for the probability of selecting a heuristic for the next iterations. Due to this mechanism, a heuristic that could previously find a good solution has a higher probability of being chosen again. The authors also avoid being stuck in a local optimum by using a simulated annealing acceptance criterion to accept a deteriorating solution with a dynamic probability. They emphasise the robustness of the approach to choose the most suitable heuristics for any specific instances. This method is quite popular in solving combinatorial optimisation problems, especially for vehicle routing problems (e.g., Ropke & Pisinger (2006*b*), Lei et al. (2011), Ribeiro & Laporte (2012), Stenger et al. (2013), Dayarian et al. (2016)).

As we discussed at the beginning of the chapter, multiple neighbourhoods in tabu search are usually searched sequentially in most algorithms in the literature. Some studies selected a neighbourhood in each iteration randomly with a certain probability. The adaptive mechanism to choose a neighbourhood in each iteration has appeared only in a few papers so far. Žulj et al. (2018) combine the adaptive large neighbourhood search with tabu search to solve an order-

batching problem. However, the authors implemented them separately for different purposes; the adaptive large neighbourhood search diversifies a current solution similar to the original idea, and then tabu search is used for intensification. Therefore, the authors did not exploit the adaptive selection of neighbourhoods in tabu search directly. To the best of our knowledge, Butsch et al. (2014) are the only ones who adopted the adaptive selection on tabu search to select a proper neighbourhood from several available choices in each iteration. Although Butsch et al. (2014) increase the weight of a neighbourhood for the next selection that can improve a specific criterion that the current solution lacks, which is different from the original idea, the authors showed that the adaptive selection on multiple neighbourhoods of tabu search works well. Therefore, it is attractive to try this on our model.

To create an adaptive mechanism to select a neighbourhood in our problem, we follow the procedure of Ropke & Pisinger (2006*a*), which weights each neighbourhood on merit-based criteria.

Let n represent a neighbourhood of the tabu search algorithm, where $n = WC$, $Switch$, $Swap$ is the neighbourhood of week centre, switching week pattern, and swapping week pattern, respectively. At the beginning of the algorithm, each neighbourhood n is assigned an equal weight of w_n. The probability of selecting the neighbourhood n, φ_n, is derived by the roulette wheel selection procedure as follows:

$$\varphi_n = \frac{w_n}{\sum_n w_n}. \tag{4.5}$$

Each neighbourhood n has π_n to collect the score for its recent performances. Whenever the neighbourhood n is selected to use in the iteration, π_n is increased by the score parameter σ_1, σ_2, or σ_3 . If it can find a new incumbent (a new best solution) of the algorithm, the score will be increased by σ_1. σ_2 is the increase for the case of finding a better solution than the current one, while σ_3 is a score for the case of an empty neighbourhood, a non-improving, or an infeasible solution. Typically, σ_1 has the highest value, while σ_2 is lower, and σ_3 is the lowest in the scoring system.

Every consecutive χ iterations, the weight of each neighbourhood is updated in the following way,

$$w_n := \begin{cases} w_n, & \text{if } \tau_n = 0, \\ w_n(1-r) + r\frac{\pi_n}{\tau_n}, & \text{otherwise,} \end{cases} \tag{4.6}$$

where τ_n is the total number of times to use the neighbourhood n during these iterations, and $r \in [0, 1]$, called the *reaction factor*, controls how much emphasis we put on the most recent performances.

Then, the score and the total number of times for each neighbourhood are reset to zero. These new weights are now used to update the probability to select a neighbourhood for the next iterations. Therefore, the probability of each neighbourhood is adjusted dynamically corresponding to its effectiveness during the search.

Regarding a tabu list for the mixed neighbourhood, the week centre neighbourhood still has its week centre tabu list T_{WC}, while both week pattern neighbourhoods share the information of forbidden pairs of a customer and a week pattern in a week pattern tabu list T_{WP}. The length of T_{WC} is similar to that for the single neighbourhood. T_{WP} has the same length as that of the single swapping week pattern neighbourhood. Since these two tabu lists are completely independent and we can select any neighbourhood during the search, we have to respect both tabu lists at the same time to avoid a cycle of solutions. The aspiration criterion also applies here. When we select the week centre neighbourhood, the allocation problem is solved to get an updated vector of week patterns. Unless we get a new best solution, i.e., the aspiration criterion applies, we have to add additional constraints to the allocation problem to prohibit any tabu in T_{WP} to be part of the optimal solution as follows.

$$g_{bp} = 0 \qquad \text{where } (b, p) \in T_{WP}. \tag{4.7}$$

In other words, if a neighbouring solution in the week centre neighbourhood fails to be a new incumbent, we solve the allocation problem again with the above constraints for the corresponding vector of week patterns to respect the week pattern tabu list. Line 5 in Algorithm

10 and Line 13 in Algorithm 11 have to be slightly modified by adding Constraints (4.7) in the allocation problem.

Similarly to the search in the week pattern neighbourhoods, we respect the week centre tabu list whenever we solve the location problem. If a neighbouring solution is not a new incumbent, we update the solution, but do not change any current week centre if the week belongs to T_{WC}, i.e., we update the new week centre by Equations (4.1) only for $w \notin T_{WC}$. This change has to be made in Line 5 of Algorithm 12 and Line 5 of Algorithm 13.

Regarding the diversification scheme in the algorithm, we select either the diversification on week centres or week patterns with equal probability. Its details are similar to the single neighbourhood cases: we change some part of either the week centres or the week patterns vector of the best neighbouring solution randomly, and then use the location-allocation heuristic to improve the quality of the solution. We also use the same criteria when applying this scheme. The diversification is carried out when a neighbourhood is empty, a neighbouring solution keeps infeasible, or the algorithm is not able to improve the incumbent.

We set up the maximum number of total iterations as the stopping criterion, which is different from the single neighbourhood cases. This criterion is typical in most of the literature on the adaptive large neighbourhood search.

A modified tabu search algorithm for the mixed neighbourhood is presented in Algorithm 14. We assume that we have an initial solution S^0. Additional parameters specifically for the mixed neighbourhood to define the adaptive selection of the neighbourhoods are presented: an initial weight value for each neighbourhood w^0, the reaction factor r, and the merit-based scores for the performance of a neighbourhood σ_1, σ_2, and σ_3. The maximum total number of sub-iterations for updating the weights and the probabilities for the neighbourhoods is χ, and the maximum number of total iterations for the algorithm is $MaxIt$. The rest of the parameters for controlling the tabu search are similar to the original tabu search algorithm.

We initialise the weight for each neighbourhood with a predetermined initial value w^0. Since the neighbourhoods have identical weight, their initial probabilities to be selected are equal. We also introduce a new counter for the total number of sub-iterations it for updating the probabilities in the adaptive mechanism, and $totalIt$ is a counter of the total number of iterations for the stopping criterion.

The algorithm selects a neighbourhood for each iteration depending on the selection probabilities. During the search in a specific neighbourhood, we have to respect both of the tabu lists, as described above. After finishing the search in the neighbourhood and updating the corresponding tabu list, the score of this neighbourhood depends on its performance, as shown in Lines 5−17. In the case of an empty neighbourhood, an infeasible solution, or a deteriorating solution, the neighbourhood gets the score of σ_3, as shown in Lines 6, 8, and 15, respectively. If the neighbourhood can derive a new incumbent, it will get a reward of σ_1 in Line 11. Otherwise, the score of the neighbourhood increases by σ_2 for at least finding a better feasible solution than the current one: see Line 13. Line 18 updates the total number of sub-iterations and total number of iterations. Whenever the number of sub-iterations reaches the limit χ, the weights and probabilities are updated in Lines 20 and 21. Then, the information of the scores, the total number of times to use each neighbourhood, and the number of sub-iterations are reinitialised. The subsequent steps are similar to the original tabu search algorithm in Algorithm 9, except in Lines 24−26 since the maximum number of consecutive iterations to run diversification is not a stopping criterion here. The neighbouring solution is assessed for updating the penalty factor and the new incumbent. Note that the diversification is slightly modified for the mixed neighbourhood. The algorithm returns the incumbent when the algorithm reaches the stopping criterion.

4.5 Data Generation and Programme Set-Up

Similarly to the experiment in the previous chapters, we test the algorithm on the same data instances in the same computer. Namely, there are 9 combinations and five data instances for each combination, so there are 45 instances in total in the experiment. As we discussed in the previous chapter, we can face computational difficulties for more complicated instances, i.e., the ones with more customers in the longer planning horizon. Since we start the tabu search

Algorithm 14 The Tabu Search Algorithm for the Mixed Neighbourhood (Mix)

Input: S^0
Parameter:
For mixed neighbourhood control : w^0, r, $\sigma_1, \sigma_2, \sigma_3$, χ, $MaxIt$
For general control : k, N_V, N_{NI}, ϕ^*
Initial:
$\overline{S}$:= S^0, S^*:= S^0, $i, j, v := 0$, $\phi = 1$,
$\pi_n := 0$, $\tau_n := 0$, $w_n := w^0$,
Calculate φ_n from (4.5) where $n \in N$
$it := 0$, $totalIt := 0$
1: **while** $totalIt < MaxIt$ **do**
2: Randomly select a neighbourhood n by using φ_n, $n \in \{WC,\ Switch,\ Swap\}$
3: $\tau_n := \tau_n + 1$ ▷ Update the total number to select a neighbourhood n
4: Find the best non-tabu solution S' in the neighbourhood n of $\overline{S}$
5: **if** the neighbourhood is empty **then**
6: $\pi_n := \pi_n + \sigma_3$
7: **else if** S' is infeasible **then**
8: $\pi_n := \pi_n + \sigma_3$
9: **else**
10: **if** S' is better than S^* **then**
11: $\pi_n := \pi_n + \sigma_1$
12: **else if** S' is better than $\overline{S}$ **then**
13: $\pi_n := \pi_n + \sigma_2$
14: **else**
15: $\pi_n := \pi_n + \sigma_3$ ▷ S' is a non-improving solution
16: **end if**
17: **end if**
18: $it := it + 1, totalIt := totalIt + 1$ ▷ Increment total number of sub-iterations and all iterations
19: **if** $it = \chi$ **then**
20: Update w_n by using (4.6) ▷ Update all information of the adaptive mechanism
21: Calculate φ_n from (4.5)
22: Reinitialise π_n:= 0, $\tau_n := 0$, $it := 0$
23: **end if**
24: Follow Lines 3−23, and 27−37 of Algorithm 9
25: **end while**
Output: S^*

algorithm with a random week centre vector, we collect five repetitions of the experiment per data instance to get more reliable statistical results. Therefore, there are 25 results per combination and 225 results in total. The value of τ^{week} is the same as in Chapter 3, which is 0.05.

Under the same programme set-up as in Chapter 3, the experiment is run on a single thread without pre-processing. Regarding the allocation problem and the location problem, we use CPLEX to solve only the allocation problem for at most 10 seconds. Since we prefer a feasible solution for an initial solution and a diversified solution in the diversification, we set the temporary high penalty factor to 100 and the maximum number of repetitions to run the location-allocation heuristic to 15.

According to the notations of the main algorithm (Algorithm 9) for a single neighbourhood, the stopping criteria, *maxTime* and N_D, are 900 seconds (15 minutes) and 5, respectively. We let k and ϕ^* be 2 and 32 for penalising infeasible solutions. We set N_V and N_{NI} to be proportional to the length of the tabu list of a neighbourhood, namely,

$$N_{NI} = 2 \cdot |T|,$$
$$N_V = \frac{1}{6} \cdot N_{NI}$$

where $|T|$ is the length of the tabu list of a single neighbourhood. Note that the length of the tabu list of the week centre neighbourhood, the switching week pattern neighbourhood, and the swapping week pattern neighbourhood are $\left\lfloor \frac{|W|}{2} \right\rfloor$, $\lfloor \frac{1}{2} \sum_{b \in B^{NW}} r_b \rfloor$, and $\lfloor \frac{1}{4} \sum_{b \in B^{NW}} r_b \rfloor$, respectively. Therefore, each neighbourhood has different values of N_V and N_{NI} corresponding to the different length of their tabu list.

For the mixed neighbourhood (Algorithm 14), N_{NI} and N_V are derived in the same way as above. In this case, $|T|$ is the average length of the tabu lists for the week centre neighbourhood and the week pattern neighbourhoods, i.e.,

$$|T| = \frac{|T_{WC}| + |T_{WP}|}{2}$$

where $|T_{WC}| = \left\lfloor \frac{|W|}{2} \right\rfloor$ and $|T_{WP}| = \lfloor \frac{1}{4} \sum_{b \in B^{NW}} r_b \rfloor$, are the lengths of the tabu lists for the week centre neighbourhood and the week pattern neighbourhoods, respectively.

Regarding the other parameters for the mixed neighbourhood, we set the initial weight w^0 at 1 for every neighbourhood. The scores σ_1, σ_2, and σ_3 are 50, 20, and 5, respectively. We update the adaptive probabilities every 50 iterations (i.e., χ is 50) and r is 0.2. Finally, the stopping criterion, *MaxIt*, is 1000 iterations.

4.6 Computational Results

In this section, we show the best results from the previous chapter and set them as the benchmark for the tabu search algorithm. Next, we compare the initial solutions from the location-allocation heuristic to the benchmark to see the quality of the solutions at the beginning of the tabu search. Then, the performance of tabu search on a single neighbourhood, i.e., the week centre and the week pattern neighbourhoods, is discussed. For each neighbourhood, we investigate the effectiveness of the aspiration criterion first, as it increases the search space by including also tabu solutions. If it cannot improve the algorithm, we might save more time by skipping all tabu solutions. Then, we study the effect of reducing the size of the neighbourhoods. In the case of the week centre neighbourhood, we further investigate the benefits of using QAP, i.e., using the surrogate objective function from Section 4.2.1 to determine the best move in each iteration.

After deciding on the best configuration for every single neighbourhood, we compare them to the mixed neighbourhood. Note that in the experiment of each data instance every algorithm is started with the same initial week centres for a fair comparison. Moreover, we investigate the improvement in the initial solution and the quality of our simple diversification. These are to provide more insights into the effectiveness of the tabu search. Then, we compare the

best results from the tabu search algorithm to the benchmark. To confirm the robustness of the tabu search, we compare the initial solutions and the best solutions found by the tabu search to heuristics in CPLEX. Finally, we combine the tabu search with our previously developed Benders' decomposition algorithm from Chapter 3 and observe the efficiency of this combination.

As we mentioned at the beginning of the chapter, we focus on the relative percentage gap from Equation (4.3) to evaluate the performance of the tabu search algorithm in each data instance, where the non-negative value indicates that the heuristic performs as well as or even better than the benchmark. Moreover, among 25 results in each combination, we count the number of results that are at least as good as those of the benchmark, i.e., their relative percentage gaps are greater than -0.01%. We define them as *comparable results*. Also, we count the number of results that are strictly better than those of the benchmark, specifically where the relative percentage gaps are higher than 0.01%. Additional information of interest is the average computational time in each combination. In some part of the analysis, we provide additional information to give more insight, for example, the number of results that can find the same optimal solutions as the benchmark.

4.6.1 Benchmark for Tabu Search

We present the information about the best-found solutions from the previous chapter in Table 4.2. For each combination, we present the best, average, and the worst relative percentage gaps between lower and upper bounds among 5 data instances in Columns *Best %Gap*, *%Gap*, and *Worst %Gap*, respectively. We found the optimal solution in some data instances and show the number of them in Column *#Opt (45)*, where the number in brackets is the total number of data instances which equals the maximum possible total of *#Opt*. In this case, there are 29 out of 45 data instances that can find the optimal solutions: see the total of *#Opt* in the last row. This information in Column *#Opt (45)* limits the maximum number of results that can be better than the benchmark, as shown in Column *Max #Better*, since we cannot get better objective values than the optimal solutions.

In some less challenging combinations such as 30_1 and 30_2 that can find the optimal solution in every data instance, it is not possible to find better results, resulting in a value of zero in Column *Max #Better*. In more complicated combinations, the exact solution approaches from Chapter 3 cannot always find the optimal solutions, so in these cases, the heuristic may find better solutions. In these combinations, Column *Max #Better* shows the highest possible total number of better results that we can achieve. For example, in 30_3, the optimal solutions for two data instances still cannot be found. Since, due to repetitions, we obtain 5 results for each data instance, there will be at most 10 results that may yield a better solution. The maximum possible total number of results that perform better than the benchmark in the experiment is presented below the column. *Total time (s)* in the last column shows the average computational time in seconds.

4.6.2 Comparison between the Initial Solutions and the Benchmark

We compare the performance of the initial solution of each result to the benchmark to see its quality. Table 4.3 focuses on the relative percentage gaps compared to the benchmark from Section 4.6.1, i.e., by using Equation (4.3). Columns *Worst %Gap**, *%Gap**, and *Best %Gap** present the worst, the average, and the best relative percentage gap among 25 results of each combination, respectively. For each combination, the number of results that are at least comparable to the benchmark is presented in Columns *#Good (225)*. Since every result can be as good as those of the benchmark, the maximum possible total of *#Good* is the total number of results, as shown in the brackets. Sub-column *Initial* of Columns *#Opt* presents the number of results whose initial solutions are the same as the optimal solutions found by the benchmark. Since the benchmark cannot always find the optimal solution, for example, in the most complicated combinations like 40_3 and 50_3, we show the maximum number of possible cases in Sub-column *Max*. Note that the number in Sub-column *Max* here is five times the corresponding number of data instances in Column *#Opt (45)* of Table 4.2 due to the repetitions. Finally, the number of results that are strictly better than the benchmark for each

Data	Best % Gap	% Gap	Worst %Gap	#Opt (45)	Max #Better	Total time (s)
30_1	0.00	0.00	0.01	5	0	6.4
30_2	0.00	0.01	0.01	5	0	120
30_3	0.01	2.79	8.29	3	10	586.2
40_1	0.00	0.01	0.01	5	0	29.6
40_2	0.01	0.01	0.01	5	0	291.2
40_3	0.01	13.20	21.14	1	20	820.6
50_1	0.01	0.01	0.01	5	0	44.2
50_2	1.04	2.42	4.89	0	25	900.8
50_3	14.09	19.55	23.20	0	25	902.4
Total				29	80	

Table 4.2: The information about the best-found solutions from the exact solution approaches in Chapter 3.

combination is presented in Sub-column *Initial* of Columns *#Better*, where each combination has the specific number of maximum possible cases as shown in Sub-column *Max*. The values in Sub-column *Max* here are obtained from Column *Max #Better* in Table 4.2. The total of *#Opt*, *#Good* and *#Better* from every combination are shown in the last row for the overall results. Note that the last row of *Max* in Columns *#Opt* and *#Better* present the maximum total of possible cases in the corresponding aspects. We do not focus on the average computational time in this section because the location-allocation heuristic spends less than one second to generate an initial solution.

In terms of relative percentage gaps, the negative values in *Worst %Gap** and *%Gap** imply that the quality of the initial solutions is overall worse than the benchmark. However, the location-allocation heuristic manages to find the same optimal solution as the benchmark, especially in some less challenging data instances, e.g., in 30_1 and 30_2. In particular, the total of *#Opt* supports that the algorithm finds the optimal solutions in 24% of the total number of possible cases. Moreover, the algorithm manages to derive comparable solutions in 20% of the total results as shown in the total of *#Good*. In more complicated instances like 40_3 and 50_3 where the benchmark struggles to find the optimal solution, the algorithm can find better solutions: see the corresponding positive values in *Best %Gap** and *#Better*.

From the above discussion, the location-allocation heuristic can derive comparable solutions and better solutions in some cases even though it spends less than one second. However, the majority of results cannot find those solutions and the overall quality of the solutions is not high. Therefore, these initial solutions still require improvement in terms of solution quality.

Next, we will discuss the effectiveness of every single neighbourhood and the mixed neighbourhood. Note that for the single neighbourhoods, we will investigate the effectiveness of the aspiration criterion and the effect of the reduced search. We still focus on the performance in terms of the solution quality and the average computational time as in this section. However, we do not focus on the number of optimal solutions since we aim for finding better solutions if possible and for such cases the information of *#Better* is already enough.

4.6.3 Experiments on the Week Centre Neighbourhood

Effectiveness of the Aspiration Criterion

Tables 4.4 and 4.5 present the effectiveness of the aspiration criterion in the tabu search algorithm with EWC (Algorithm 10), where *EWC* and *EWC+* represent the algorithm without and with the aspiration criterion, respectively. Similarly in Section 4.6.2, we show the information of the relative percentage gaps compared to benchmark in *Worst %Gap**, *%Gap**, and *Best %Gap**. Also, we count the number of comparable results and better results, and present them in Columns *#Good (225)* and *#Better*, respectively. Again, the maximum total of possible

Data	Worst %Gap*	%Gap*	Best %Gap*	#Good (225)	#Opt		#Better	
					Initial	Max	Initial	Max
30_1	-5.56	-1.08	0.00	11	11	25	0	0
30_2	-5.47	-1.06	0.00	12	12	25	0	0
30_3	-16.40	-4.91	-0.50	0	0	15	0	10
40_1	-7.35	-1.24	0.00	1	1	25	0	0
40_2	-2.69	-0.83	0.00	6	6	25	0	0
40_3	-14.69	-3.01	2.64	4	1	5	2	20
50_1	-10.72	-2.12	0.00	4	4	25	0	0
50_2	-2.57	-1.07	0.29	2	0	0	1	25
50_3	-6.82	-2.09	2.85	5	0	0	5	25
Total				45	35	145	8	80

Table 4.3: The performance of the initial solution from the location-allocation heuristic compared to the benchmark.

cases for *#Good* is shown in the brackets, i.e., the total number of results. Similarly to the previous section, for *#Better*, each combination has a specific number of possible cases shown in Sub-column *Max*. The last row of *#Good* and *#Better* present the total corresponding results.

Table 4.4 shows that both algorithms are not significantly different in terms of the relative percentage gaps. However, the algorithm without the aspiration criterion tends to have a larger number of *#Good* and *#Better* in more complicated combinations, especially for 50 customers, resulting in the higher total of *#Good* and *#Better*.

Moreover, *EWC* spends less average time: see Columns *Total time (s)* in Table 4.5. These results indicate that the aspiration criterion does not help to improve the algorithm, so the algorithm without the aspiration criterion is preferable.

Nevertheless, Columns *Time/Iteration (s)* of Table 4.5 show that the exhaustive search in the week centre neighbourhood, even in the case without the aspiration criterion, spends a significant amount of computational time per iteration on average, especially in the most difficult combinations like 30_3, 40_3, and 50_3. Therefore, the algorithm still requires more techniques to speed up the search.

Comparison between Different Size Reductions on the Neighbourhood

To speed up the tabu algorithm on the week centre neighbourhood without the aspiration criterion, we reduce the size of the neighbourhood with different values of the parameter α. Here, α is 0.5, 0.3, and 0.1, i.e., we investigate only the best 50%, 30%, and 10% of total customers qualifying from the measurement (4.4) in Section 4.2.1. Tables 4.6, 4.7, and 4.8 show the performance on the same criteria, i.e., the relative percentage gaps and the average computational time, where *EWC* and α-*WC* represent the exhaustive search and the reduced search corresponding to α, respectively.

0.1-WC tends to have better *%Gap** and *Best %Gap** than the other algorithms when the data instances become more challenging, as shown in Table 4.6. Although Table 4.7 shows that *0.1-WC* has the worst total of *#Good*, this is a result of the poor performances in the less challenging combinations (30_1, 40_1, 40_2, 50_1, and 50_2). When we restrict our attention to the more challenging combinations such as 40_3 and 50_3, *0.1-WC* has better *#Good* and *#Better* than the other algorithms. More importantly, it has the highest total of *#Better*.

Furthermore, it is clear that a smaller value of α results in a better average computational time, since there are fewer customers to investigate per iteration, as shown in Column *Total time (s)* in Table 4.8. *0.1-WC* can improve the computational time of the exhaustive search by at least 75% in the less difficult combinations, e.g., 30_1, 40_1, and 50_1. For the most challenging combinations, i.e., 30_3, 40_3, and 50_3, *0.1-WC* spends 30%−50% less time, which is still a noticeable amount.

Due to the significant reduction of time and the superior performance for the most difficult

Data	Worst %Gap*		%Gap*		Best %Gap*		#Good (225)		#Better		
	EWC	**EWC+**	**EWC**	**EWC+**	**EWC**	**EWC+**	**EWC**	**EWC+**	**EWC**	**EWC+**	**Max**
30_1	-0.10	0.00	0.00	0.00	0.00	0.00	24	25	0	0	0
30_2	0.00	0.00	0.00	0.00	0.00	0.00	25	25	0	0	0
30_3	-6.37	-6.37	-2.08	-1.86	0.00	0.00	6	6	0	0	10
40_1	0.00	-0.05	0.00	0.00	0.00	0.00	25	24	0	0	0
40_2	-0.01	0.00	0.00	0.00	0.00	0.00	24	25	0	0	0
40_3	-6.55	-7.43	-1.01	-1.32	4.16	4.16	7	8	5	6	20
50_1	-0.12	-0.12	0.00	-0.01	0.00	0.00	24	22	0	0	0
50_2	0.00	0.00	0.06	0.06	0.29	0.29	25	25	15	15	25
50_3	-6.01	-6.01	-0.04	-0.36	2.85	2.85	13	8	13	8	25
Total							173	168	33	29	80

Table 4.4: The effectiveness of the aspiration criterion on tabu search with EWC (1).

Data	Total time (s)		Time/Iteration (s)	
	EWC	**EWC+**	**EWC**	**EWC+**
30_1	68.32	81.68	1.26	1.60
30_2	388.64	463.72	4.52	5.76
30_3	993.64	970.36	158.43	156.87
40_1	165.08	202.84	2.75	3.43
40_2	486.72	617.52	5.98	7.88
40_3	973.60	974.64	143.11	151.00
50_1	185.20	226.72	3.21	4.17
50_2	700.76	814.80	7.74	10.48
50_3	921.40	936.28	52.27	66.29

Table 4.5: The effectiveness of the aspiration criterion on tabu search with EWC (2).

data instances, the best version of the week centre neighbourhood now is the 0.1-reduce-size neighbourhood without the aspiration criterion.

However, when we observe the average time per iteration in the reduced-size neighbourhoods in Columns *Time/Iteration (s)* of Table 4.8, *0.1-WC* still spends quite a long time per iteration in more complicated combinations. This situation stems from solving the allocation problem, which is a mixed-integer programme for every move operation. Therefore, we will save the computational time by using QAP (Quick Allocation Problem), which uses the surrogate objective value for the allocation problem during the search.

Outstanding Performances of the QAP

Tables 4.9 and 4.10 show the comparison of the performance between *0.1-WC* which is the best one so far and Algorithm 11 (EWC-QAP). The latter algorithm is represented by *EWC** in the tables.

In Table 4.9, both algorithms are competitive in terms of the relative percentage gaps. Columns *#Good (225)* and *#Better* provide more insights; *EWC** has the better total of *#Good* and that of *#Better*. Therefore, *EWC** has the better solution quality.

Regarding the average computational time in Columns *Total time (s)* in Table 4.10, *EWC** significantly outperforms the other algorithm; it spends at least 90% less time than *0.1-WC* in every combination. This is because *EWC** finishes the search really quickly, i.e., less than 1 second, in every iteration, as shown in Column *Time/Iteration (s)* of Table 4.10.

The above results support that EWC-QAP is more effective while requiring much less computational time.

Next, we investigate further if the reduced-size neighbourhood can enhance QAP. Tables 4.11-4.13 show the comparison between QAP with the exhaustive search and with the reduced-size neighbourhood by the parameter α, represented by *EWC** and α-*WC**, respectively. We use the same values of α as before which are 0.5, 0.3, and 0.1.

The results in Table 4.12 show that the reduced-size algorithms have worse performance in terms of the total of *#Good* and *#Better*. This implies that the size reduction does not enhance the effectiveness of QAP. Furthermore, the reduced-size neighbourhood does not help to reduce the computational time, as every algorithm spends a similar amount of time on average in every combination: see Table 4.13. Therefore, QAP is effective and efficient without having to reduce the size of the neighbourhood.

4.6.4 Experiments on the Switching Week Pattern Neighbourhood

Effectiveness of the Aspiration Criterion

Regarding the switching week pattern neighbourhood, we investigate the effectiveness of the aspiration criterion first. Table 4.14 and 4.15 show the comparison of the results without and with the aspiration criterion, represented by *Switch* and *Switch+*, respectively.

Data	Worst %Gap*				%Gap*				Best %Gap*			
	EWC	**0.5-WC**	**0.3-WC**	**0.1-WC**	**EWC**	**0.5-WC**	**0.3-WC**	**0.1-WC**	**EWC**	**0.5-WC**	**0.3-WC**	**0.1-WC**
30_1	-0.10	0.00	0.00	-0.27	0.00	0.00	0.00	-0.03	0.00	0.00	0.00	0.00
30_2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30_3	-6.37	-6.37	-6.37	-5.81	-2.08	-1.57	-1.75	-1.09	0.00	0.00	0.47	2.42
40_1	0.00	0.00	0.00	-0.28	0.00	0.00	0.00	-0.02	0.00	0.00	0.00	0.00
40_2	-0.01	-0.01	0.00	-0.35	0.00	0.00	0.00	-0.01	0.00	0.00	0.00	0.00
40_3	-6.55	-3.21	-3.21	-5.18	-1.01	-0.20	-0.08	0.35	4.16	4.16	4.01	4.95
50_1	-0.12	-0.12	-0.12	-0.12	0.00	-0.01	0.00	-0.01	0.00	0.00	0.00	0.00
50_2	0.00	0.00	0.00	-0.59	0.06	0.06	0.06	0.00	0.29	0.29	0.29	0.29
50_3	-6.01	-1.91	-1.83	-1.14	-0.04	0.58	0.76	1.01	2.85	2.85	2.85	2.85

Table 4.6: The performance of the different size reductions in the week centre neighbourhood (1).

Data	#Good (225)				#Better				
	EWC	**0.5-WC**	**0.3-WC**	**0.1-WC**	**EWC**	**0.5-WC**	**0.3-WC**	**0.1-WC**	**Max**
30_1	24	25	25	20	0	0	0	0	0
30_2	25	25	25	25	0	0	0	0	0
30_3	6	8	9	8	0	0	1	1	10
40_1	25	25	25	22	0	0	0	0	0
40_2	24	23	25	22	0	0	0	0	0
40_3	7	11	11	14	5	8	10	11	20
50_1	24	23	24	22	0	0	0	0	0
50_2	25	25	25	20	15	15	15	11	25
50_3	13	16	15	19	13	16	15	19	25
Total	173	181	184	172	33	39	41	42	80

Table 4.7: The performance of the different size reductions in the week centre neighbourhood (2).

Data	Total time (s)				Time/Iteration (s)			
	EWC	**0.5-WC**	**0.3-WC**	**0.1-WC**	**EWC**	**0.5-WC**	**0.3-WC**	**0.1-WC**
30_1	68.32	37.16	31.44	15.96	1.26	0.72	0.54	0.30
30_2	388.64	177.92	113.00	53.32	4.52	2.10	1.35	0.65
30_3	993.64	898.68	804.80	511.16	158.43	65.61	43.39	10.22
40_1	165.08	82.56	58.48	29.16	2.75	1.43	1.04	0.50
40_2	486.72	211.80	143.28	60.32	5.98	2.65	1.71	0.74
40_3	973.60	927.56	905.84	681.00	143.11	54.96	39.75	11.10
50_1	185.20	99.16	60.48	32.64	3.21	1.72	1.15	0.57
50_2	700.76	332.00	186.60	86.28	7.74	3.72	2.14	1.01
50_3	921.40	910.40	835.96	466.96	52.27	23.15	13.19	4.01

Table 4.8: The performance of the different size reductions in the week centre neighbourhood (3).

Data	Worst %Gap*		%Gap*		Best %Gap*		#Good (225)		#Better		
	0.1-WC	**EWC***	**0.1-WC**	**EWC***	**0.1-WC**	**EWC***	**0.1-WC**	**EWC***	**0.1-WC**	**EWC***	**Max**
30_1	-0.27	0.00	-0.03	0.00	0.00	0.00	20	25	0	0	0
30_2	0.00	-2.09	0.00	-0.08	0.00	0.00	25	24	0	0	0
30_3	-5.81	-4.16	-1.09	-1.44	2.42	2.08	8	4	1	1	10
40_1	-0.28	-0.20	-0.02	-0.01	0.00	0.00	22	22	0	0	0
40_2	-0.35	-0.08	-0.01	0.00	0.00	0.00	22	23	0	0	0
40_3	-5.18	-5.18	0.35	0.52	4.95	4.16	14	17	11	14	20
50_1	-0.12	-0.13	-0.01	-0.02	0.00	0.00	22	20	0	0	0
50_2	-0.59	0.00	0.00	0.06	0.29	0.29	20	25	11	10	25
50_3	-1.14	-0.87	1.01	1.11	2.85	2.85	19	19	19	19	25
Total							172	179	42	44	80

Table 4.9: The performance of tabu search with 0.1-WC and EWC-QAP (1).

Data	Total time (s)		Time/iteration (s)	
	0.1-WC	**EWC***	**0.1-WC**	**EWC***
30_1	15.96	0.88	0.30	0.02
30_2	53.32	1.04	0.65	0.01
30_3	511.16	14.56	10.22	0.19
40_1	29.16	2.72	0.50	0.04
40_2	60.32	2.4	0.74	0.03
40_3	681.00	12.8	11.10	0.15
50_1	32.64	1.44	0.57	0.03
50_2	86.28	2.56	1.01	0.03
50_3	466.96	12.36	4.01	0.09

Table 4.10: The performance of tabu search with 0.1-WC and EWC-QAP (2).

Table 4.14 shows that there is no significant difference in terms of the relative percentage gaps between both algorithms. However, the aspiration criterion helps the algorithm to obtain a larger number of results that are as good as or better than the benchmark, especially when the data instances become more complicated: see Columns *#Good (225)* and *#Better* in Table 4.15. As a result, *Switch+* has the higher total of *#Good* and that of *#Better*.

Regarding the average computational time in Columns *Total time (s)* of Table 4.15, the algorithm with the aspiration criterion slightly increases the computational time on average. However, the overall computational time of the algorithm is still quick.

Although the aspiration criterion marginally increases time in the algorithm, this is acceptable compensation for achieving better results. Therefore, the aspiration criterion is beneficial for the switching neighbourhood.

Comparison between Different Size Reductions on the Neighbourhood

Next, we study the benefits of reducing the neighbourhood size, where the results of both algorithms are shown in Tables 4.16−4.18.

In terms of the relative percentage gaps in Table 4.16, each algorithm has similar results in *%Gap** and *Best %Gap**. Table 4.17 provides more explicit insightful information: the reduced size of the neighbourhood can worsen the overall quality of the search: note the smaller total of *#Good* and that of *#Better* in every reduced-size algorithm. In particular, the total of *#Good* in those algorithms drops by at least 5%, compared to the exhaustive search.

Regarding the average computational time in Table 4.18, the reduced-size neighbourhoods decrease the computational time on average. Therefore, *Switch+* spends the longest computational time. However, overall it is still quick enough.

Since we focus more on the quality of the search here, we conclude that the size reduction is not helpful enough for the switching neighbourhood.

4.6.5 Experiments on the Swapping Week Pattern Neighbourhood

Effectiveness of the Aspiration Criterion

In the case of the swapping week pattern neighbourhood, we investigate it in the same way. First, the performance of the aspiration criterion in the neighbourhood is presented in Tables 4.19 and 4.20, where *Swap* and *Swap+* denote the neighbourhood without and with the aspiration criterion, respectively.

In Table 4.19, both algorithms do not exhibit considerable differences in terms of the relative percentage gaps. Table 4.20 provides more insights showing that the algorithm with the aspiration criterion has the higher total of *#Good* and *#Better*. The aspiration criterion, however, slightly increases the computational time on average: see Columns *Total time (s)* in Table 4.20.

Although the aspiration criterion tends to require a marginally larger amount of computational time, this is acceptable because overall it increases the solution quality. Therefore, we

Data	Worst %Gap*				%Gap*				Best %Gap*			
	EWC*	**0.5-WC***	**0.3-WC***	**0.1-WC***	**EWC***	**0.5-WC***	**0.3-WC***	**0.1-WC***	**EWC***	**0.5-WC***	**0.3-WC***	**0.1-WC***
30_1	0.00	-0.10	0.00	-2.15	0.00	0.00	0.00	-0.10	0.00	0.00	0.00	0.00
30_2	-2.09	0.00	0.00	0.00	-0.08	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30_3	-4.16	-5.96	-5.17	-4.87	-1.44	-1.55	-1.13	-1.35	2.08	2.42	2.42	2.42
40_1	-0.20	-0.19	-0.10	-0.20	-0.01	-0.02	-0.01	-0.04	0.00	0.00	0.00	0.00
40_2	-0.08	-0.08	-0.08	-0.25	0.00	-0.01	-0.01	-0.02	0.00	0.00	0.00	0.00
40_3	-5.18	-5.18	-1.65	-1.89	0.52	0.23	0.52	0.44	4.16	4.16	4.95	4.95
50_1	-0.13	-0.13	-0.13	-1.92	-0.02	-0.02	-0.01	-0.10	0.00	0.00	0.00	0.00
50_2	0.00	-0.11	0.00	-0.55	0.06	0.05	0.06	0.01	0.29	0.29	0.29	0.29
50_3	-0.87	-2.93	-1.02	-1.02	1.11	0.79	1.02	0.92	2.85	2.85	2.85	2.85

Table 4.11: The comparison of different size reductions on tabu search with QAP (1).

Data	#Good (225)				#Better				
	EWC*	**0.5-WC***	**0.3-WC***	**0.1-WC***	**EWC***	**0.5-WC***	**0.3-WC***	**0.1-WC***	**Max**
30_1	25	24	25	20	0	0	0	0	0
30_2	24	25	25	25	0	0	0	0	0
30_3	4	6	8	4	1	3	2	1	10
40_1	22	21	20	15	0	0	0	0	0
40_2	23	21	22	21	0	0	0	0	0
40_3	17	13	10	12	14	10	8	10	20
50_1	20	20	22	18	0	0	0	0	0
50_2	25	22	25	20	10	9	9	10	25
50_3	19	16	18	20	19	16	18	20	25
Total	179	168	175	155	44	38	37	41	80

Table 4.12: The comparison of different size reductions on tabu search with QAP (2).

Data	Total time (s)			
	EWC*	**0.5-WC***	**0.3-WC***	**0.1-WC***
30_1	0.88	0.88	0.84	0.8
30_2	1.04	1.28	1.24	1.28
30_3	14.56	15.96	15.2	16.24
40_1	2.72	2.8	2.6	2.6
40_2	2.4	2.32	2.4	2.56
40_3	12.8	11.08	12	10.92
50_1	1.44	1.4	1.4	1.6
50_2	2.56	2.72	2.08	2.52
50_3	12.36	11.12	12.68	12.16

Table 4.13: The comparison of different size reductions on tabu search with QAP (3).

Data	Worst %Gap*		%Gap*		Best %Gap*	
	Switch	**Switch+**	**Switch**	**Switch+**	**Switch**	**Switch+**
30_1	-2.77	-2.74	-0.47	-0.14	0.00	0.00
30_2	-1.70	-1.18	-0.07	-0.11	0.00	0.00
30_3	-3.03	-4.96	-0.89	-1.45	2.42	2.08
40_1	-0.28	-0.30	-0.06	-0.06	0.00	0.00
40_2	-1.04	-1.09	-0.10	-0.06	0.00	0.00
40_3	-2.32	-1.76	0.64	0.91	4.14	4.95
50_1	-1.21	-0.32	-0.14	-0.06	0.00	0.00
50_2	-1.75	-0.55	-0.11	-0.02	0.29	0.29
50_3	-4.61	-1.03	0.66	0.88	2.85	2.85

Table 4.14: The effectiveness of the aspiration criterion on tabu search with Switch (1).

Data	#Good (225)		#Better			Total time (s)	
	Switch	**Switch+**	**Switch**	**Switch+**	**Max**	**Switch**	**Switch+**
30_1	14	19	0	0	0	0.88	1.20
30_2	24	22	0	0	0	1.00	1.28
30_3	3	2	1	1	10	8.60	7.92
40_1	12	12	0	0	0	2.92	3.40
40_2	15	18	0	0	0	2.64	3.16
40_3	13	14	11	13	20	8.52	9.12
50_1	16	16	0	0	0	4.56	6.52
50_2	14	15	5	7	25	4.16	5.76
50_3	18	18	18	18	25	10.12	11.00
Total	129	136	35	39	80		

Table 4.15: The effectiveness of the aspiration criterion on tabu search with Switch (2).

prefer the swapping week pattern neighbourhood with the aspiration criterion.

Comparison between Different Size Reductions on the Neighbourhood

Now we further investigate the benefits of limiting the search in the swapping week pattern neighbourhood, as shown in Tables 4.21− 4.23. We reduce the size of the neighbourhood with the corresponding parameter $\alpha \in \{0.5, 0.3, 0.1\}$ and call it α-*Swap+*.

The performance in terms of the relative percentage gaps of each algorithm is compared in Table 4.21. Table 4.22 shows the comparison in terms of the number of results that perform well or even better than the benchmark. In terms of *#Good*, *Switch+* performs best: see the highest total of *#Good*. *0.5-Swap+* and *0.3-Swap+* come the second-best with around the 3% smaller total of *#Good* than the exhaustive search. *0.1-Swap+* is the worst in this aspect; it tends to miss several good solutions in the less complicated combinations, for example, 30_1, 30_2, and 40_2, resulting in the 14% smaller total of *#Good*. In terms of *#Better*, every algorithm has the same total of *#Better*.

Regarding the average computational time in Table 4.23, the smaller size of the neighbourhood clearly results in less computational time on average. However, as discussed above, the reducing size of the neighbourhood worsens the search quality to get comparable solutions. Therefore, we have to find a balance between these two aspects.

0.1-Swap+ is the quickest algorithm but it has the worst total number of *#Good*. *0.3-Swap+* comes second in terms of the computational time; it improves the average computational time of the exhaustive search by 34% and 46% in 40_3 and 50_3, which are the most complicated combinations in the experiments. Although its total of *#Good* is four cases fewer than that of the exhaustive search, it is quite competitive and tends to perform well in the most complicated combinations. Therefore, we conclude that *0.3-Swap+* has the best balance, i.e., it is the best configuration for the swapping neighbourhood.

4.6.6 Experiments on the Mixed Neighbourhood

Now, we have found the best configuration for every single neighbourhood under the same standard as summarised in Table 4.24. Namely, they are the week centre neighbourhood with QAP, the switching neighbourhood with the aspiration criterion, and the swapping neighbourhood with the aspiration criterion and 30% reduced-size neighbourhood.

If we focus on the performance of the single neighbourhoods, overall *EWC** has the best solution quality, while *Switch+* and *0.3-Swap+* has poorer performance in this aspect. These observations are implied from the corresponding total of *#Good* and that of *#Better* in Tables 4.12, 4.17, and 4.22. Figure 4.1 shows the performance of every single neighbourhood in data instances of the most challenging combinations like 30_3, 40_3, and 50_3 to confirm the observations. Note that every line graph represents the trend of the relative percentage gap between

Data	Worst %Gap*				%Gap*				Best %Gap*			
	Switch+	**0.5-Switch+**	**0.3-Switch+**	**0.1-Switch+**	**Switch+**	**0.5-Switch+**	**0.3-Switch+**	**0.1-Switch+**	**Switch+**	**0.5-Switch+**	**0.3-Switch+**	**0.1-Switch+**
30_1	-2.74	-2.89	-5.56	-2.15	-0.14	-0.21	-0.80	-0.17	0.00	0.00	0.00	0.00
30_2	-1.18	-0.42	-1.98	-0.42	-0.11	-0.07	-0.10	-0.02	0.00	0.00	0.00	0.00
30_3	-4.96	-2.77	-5.17	-5.96	-1.45	-1.11	-1.12	-1.10	2.08	2.08	2.42	2.08
40_1	-0.30	-1.51	-2.82	-1.35	-0.06	-0.24	-0.30	-0.17	0.00	0.00	0.00	0.00
40_2	-1.09	-1.08	-1.09	-0.14	-0.06	-0.08	-0.07	-0.02	0.00	0.00	0.00	0.00
40_3	-1.76	-2.01	-3.29	-1.07	0.91	0.69	0.50	0.83	4.95	4.95	4.95	4.95
50_1	-0.32	-4.29	-1.91	-2.82	-0.06	-0.35	-0.19	-0.21	0.00	0.00	0.00	0.00
50_2	-0.55	-1.37	-0.55	-0.55	-0.02	-0.05	-0.05	-0.06	0.29	0.29	0.29	0.29
50_3	-1.03	-4.42	-2.70	-1.57	0.88	0.61	0.56	0.79	2.85	2.85	2.85	2.85

Table 4.16: The performance of the different size reductions on tabu search with Switch and the aspiration criterion (1).

Data	#Good (225)				#Better				
	Switch+	0.5-Switch+	0.3-Switch+	0.1-Switch+	Switch+	0.5-Switch+	0.3-Switch+	0.1-Switch+	Max
30_1	19	16	13	19	0	0	0	0	0
30_2	22	21	20	24	0	0	0	0	0
30_3	2	3	3	6	1	1	2	2	10
40_1	12	10	8	6	0	0	0	0	0
40_2	18	15	18	19	0	0	0	0	0
40_3	14	16	12	14	13	12	10	12	20
50_1	16	13	13	11	0	0	0	0	0
50_2	15	17	15	13	7	6	6	6	25
50_3	18	17	18	17	18	17	17	17	25
Total	136	128	120	129	39	36	35	37	80

Table 4.17: The performance of the different size reductions on tabu search with Switch and the aspiration criterion (2).

Data	Total time (s)			
	Switch+	0.5-Switch+	0.3-Switch+	0.1-Switch+
30_1	1.20	0.84	0.64	0.52
30_2	1.28	1	1	0.92
30_3	7.92	8.52	7.16	7.28
40_1	3.40	2.44	2	1.8
40_2	3.16	2.24	2.28	1.84
40_3	9.12	7.72	7.36	7.28
50_1	6.52	3.8	3.28	2.68
50_2	5.76	4.08	3.4	2.92
50_3	11.00	10.08	8.32	6.52

Table 4.18: The performance of the different size reductions on tabu search with Switch and the aspiration criterion (3).

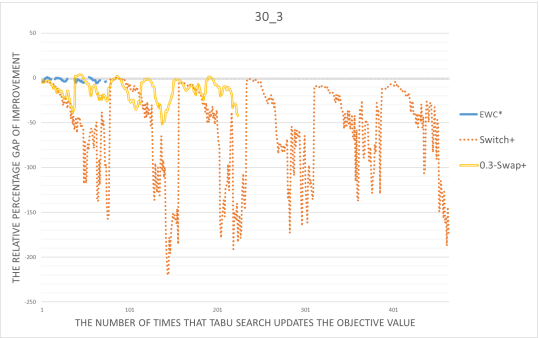
Data	Worst %Gap*		%Gap*		Best %Gap*	
	Swap	Swap+	Swap	Swap+	Swap	Swap+
30_1	-2.31	-2.90	-0.23	-0.46	0.00	0.00
30_2	-0.42	-0.63	-0.03	-0.03	0.00	0.00
30_3	-3.97	-3.87	-0.36	-0.64	2.42	2.42
40_1	-3.04	-0.88	-0.31	-0.14	0.00	0.00
40_2	-0.51	-0.08	-0.02	0.00	0.00	0.00
40_3	-1.45	-1.46	0.81	0.90	4.95	4.95
50_1	-1.92	-1.92	-0.15	-0.20	0.00	0.00
50_2	-0.23	-0.39	0.03	-0.01	0.29	0.29
50_3	-0.83	-0.93	1.01	0.92	2.85	2.85

Table 4.19: The effectiveness of the aspiration criterion on tabu search with Swap (1).

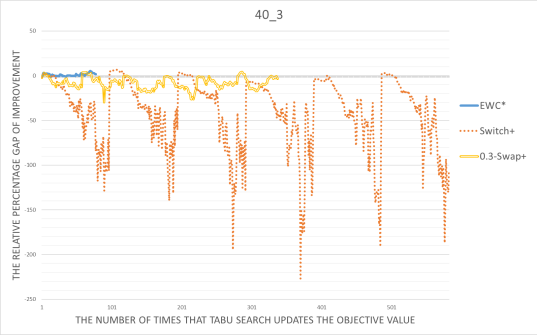
Data	#Good (225)		#Better			Total time (s)	
	Swap	Swap+	Swap	Swap+	Max	Swap	Swap+
30_1	18	16	0	0	0	0.04	0.16
30_2	23	24	0	0	0	0.72	1.04
30_3	10	8	3	3	10	9.00	9.96
40_1	3	8	0	0	0	1.88	2.72
40_2	21	22	0	0	0	2.68	3.76
40_3	14	15	13	14	20	10.36	11.92
50_1	14	15	0	0	0	3.88	5.96
50_2	20	18	5	6	25	4.76	8.16
50_3	19	18	19	18	25	14.88	19.84
Total	142	144	40	41	80		

Table 4.20: The effectiveness of the aspiration criterion on tabu search with Swap (2).

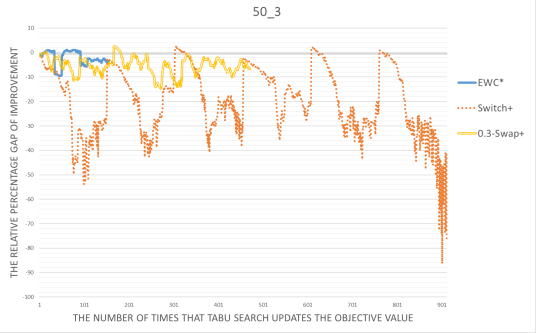
the objective value of the algorithm and the benchmark.



(a) Data instance for 30_3.



(b) Data instance for 40_3.



(c) Data instance for 50_3.

Figure 4.1: The performance of the single neighbourhoods in some complicated data instances.

However, the average computational time of the week pattern neighbourhoods for the most complicated combinations, such as 30_3, 40_3, and 50_3, is better. In particular, in such combinations, the week pattern neighbourhoods spend less than 12 seconds, while the week centre neighbourhood requires more than that: see the corresponding information in Columns *Total time (s)* of Table 4.13, 4.18, and 4.23.

We can see that each neighbourhood has its strength, so it must be beneficial to combine them into one algorithm. We then use these best configurations in the mixed neighbourhood. Tables 4.25−4.27 present the comparison of the performance between the single neighbourhoods and the mixed neighbourhood, where *Mix* is the mixed neighbourhood.

When we combine every single neighbourhood to exploit their specific search trajectories, we

Data	Worst %Gap*				%Gap*				Best %Gap*			
	Swap+	**0.5-Swap+**	**0.3-Swap+**	**0.1-Swap+**	**Swap+**	**0.5-Swap+**	**0.3-Swap+**	**0.1-Swap+**	**Swap+**	**0.5-Swap+**	**0.3-Swap+**	**0.1-Swap+**
30_1	-2.90	-3.64	-3.64	-3.87	-0.46	-0.36	-0.35	-0.72	0.00	0.00	0.00	0.00
30_2	-0.63	-1.70	-0.63	-1.18	-0.03	-0.13	-0.06	-0.10	0.00	0.00	0.00	0.00
30_3	-3.87	-2.66	-3.39	-5.17	-0.64	-0.87	-1.06	-1.12	2.42	2.42	2.08	2.08
40_1	-0.88	-0.30	-1.54	-0.31	-0.14	-0.09	-0.15	-0.09	0.00	0.00	0.00	0.00
40_2	-0.08	-0.25	-0.44	-0.25	0.00	-0.03	-0.03	-0.03	0.00	0.00	0.00	0.00
40_3	-1.46	-1.39	-0.49	-1.22	0.90	0.76	0.93	1.01	4.95	4.95	4.16	4.95
50_1	-1.92	-0.96	-0.88	-1.92	-0.20	-0.12	-0.13	-0.20	0.00	0.00	0.00	0.00
50_2	-0.39	-0.23	-0.42	-0.38	-0.01	0.05	0.00	-0.02	0.29	0.29	0.29	0.29
50_3	-0.93	-0.72	-0.52	-2.73	0.92	1.10	1.05	0.79	2.85	2.85	2.85	2.85

Table 4.21: The performance of the different size reductions on tabu search with Swap and the aspiration criterion (1).

Data	#Good (225)				#Better				
	Swap+	**0.5-Swap+**	**0.3-Swap+**	**0.1-Swap+**	**Swap+**	**0.5-Swap+**	**0.3-Swap+**	**0.1-Swap+**	**Max**
30_1	16	16	17	13	0	0	0	0	0
30_2	24	19	22	20	0	0	0	0	0
30_3	8	6	5	4	3	1	1	1	10
40_1	8	5	7	5	0	0	0	0	0
40_2	22	17	20	16	0	0	0	0	0
40_3	15	16	18	18	14	11	13	14	20
50_1	15	17	13	13	0	0	0	0	0
50_2	18	22	17	12	6	6	6	5	25
50_3	18	22	21	22	18	22	21	21	25
Total	144	140	140	123	41	40	41	41	80

Table 4.22: The performance of the different size reductions on tabu search with Swap and the aspiration criterion (2).

Data	Total time (s)			
	Swap+	**0.5-Swap+**	**0.3-Swap+**	**0.1-Swap+**
30_1	0.16	0.08	0	0
30_2	1.04	0.76	0.44	0.28
30_3	9.96	9.32	7.44	7.88
40_1	2.72	1.8	1.28	0.92
40_2	3.76	2.32	1.96	1.2
40_3	11.92	8.2	7.92	7.16
50_1	5.96	3.56	2.56	1.8
50_2	8.16	4.44	3.28	2.04
50_3	19.84	12.6	10.68	6.24

Table 4.23: The performance of the different size reductions on tabu search with Swap and the aspiration criterion (3).

Neighbourhood	The best configuration
Week centre	QAP.
Switching week pattern	Aspiration criterion.
Swapping week pattern	Aspiration criterion with 30% reduced-size neighbourhood.

Table 4.24: Summation of the best configuration for each single neighbourhood.

can see a noticeable improvement in terms of the relative percentage gaps in every combination. Table 4.25 shows that *Mix* has the best *Worst %Gap** and *Best %Gap**. The non-negative values of *%Gap** in every combination indicate that the method tends to find solutions that are at least as good as the benchmark. Table 4.26 confirms the high quality of search in the mixed neighbourhood; *Mix* has the highest number of *#Good* in every combination, especially in 30_3 where every single neighbourhood struggles to find good solutions. In particular, in most combinations, almost every result of *Mix* has a comparable solution. Similarly, in terms of *#Better*, *Mix* is superior to the other algorithms, resulting in the highest total of *#Better*.

In terms of the average computational time, Table 4.27 shows that *Mix* spends a much longer time in the algorithm. However, the mixed neighbourhood applies a different stopping criterion which is the maximum number of iterations, not the maximum computational time or the maximum number of consecutive diversifications without improving an incumbent as in the single neighbourhoods. To get a fair comparison, we check the amount of time per iteration in Columns *Time/Iteration (s)*. The results show that *Mix* spends less time in each iteration than *EWC** in general.

Due to the overall performance that we have discussed so far, the mixed neighbourhood performs a high-quality search within a reasonable amount of time and becomes the best algorithm so far.

4.6.7 Effectiveness of the Diversification in the Tabu Search

In our algorithm, the week centre neighbourhood and the week pattern neighbourhoods have different diversification schemes, while the mixed neighbourhood applies the mixture of both schemes. According to Gendreau & Potvin (2019), diversification is crucial to the effectiveness of tabu search because if it does not diversify solutions properly, it is usually the main reason for the algorithm failures. The diversification is effective if it manages to force the search to check regions of the solution space that have not been explored yet. Therefore, in this section, we use the number of results that can find a new incumbent after the first diversification as an indicator of the effectiveness. Note that each combination has a limited number of those results since the initial solutions are the optimal solutions in some cases as shown in Sub-column *Initial* of Columns *#Opt* in Table 4.3. In particular, the limit in each combination is derived from the number of results out of 25 whose initial solution is not the optimal solution and still allows to be further improved.

Table 4.28 presents the effectiveness of the diversification in every single neighbourhood and the mixed neighbourhood. Columns *EWC**, *Switch+*, *0.3-Swap+*, and *Mix* represent, respectively, the corresponding information of the week centre neighbourhood with QAP, the switching neighbourhood with the aspiration criterion, the swapping neighbourhood with the aspiration criterion and 30% reduced-size neighbourhood, and the mixed neighbourhood. Column *Max* represents the maximum possible cases in each combination that the initial solutions can get improved. The last row shows the corresponding total number of the results.

Overall, the diversification scheme for the centre neighbourhood works well; 84% of the total number of possible cases find a new incumbent after applying the diversification. For the week pattern neighbourhoods, the diversification also works effectively in around 80% of the total number of possible cases. *Mix* yields the least number of total results that get the benefit from the diversification, i.e., 77% of the total number of possible cases which, however, is still a very good amount. Therefore, we conclude that the diversification schemes that we proposed work effectively to find better solutions.

4.6.8 Improvement on the Initial Solutions by the Tabu Search

We have concluded that the mixed neighbourhood performs best in the tabu search algorithm. Here, we will show how much the algorithm can improve the initial solutions from the location-allocation heuristic. This is to prove the effectiveness of the tabu search for being capable of improving a solution.

Table 4.29 presents the information on the improvements in terms of the relative percentage gaps compared to the initial solutions. In particular, we are interested in the average and the best relative percentage gap as shown in Columns *%Gap** and *Best %Gap**. Also, we show the

Data	Worst %Gap*				%Gap*				Best %Gap*			
	EWC*	**Switch+**	**0.3-Swap+**	**Mix**	**EWC***	**Switch+**	**0.3-Swap+**	**Mix**	**EWC***	**Switch+**	**0.3-Swap+**	**Mix**
30_1	0.00	-2.74	-3.64	0.00	0.00	-0.14	-0.35	0.00	0.00	0.00	0.00	0.00
30_2	-2.09	-1.18	-0.63	0.00	-0.08	-0.11	-0.06	0.00	0.00	0.00	0.00	0.00
30_3	-4.16	-4.96	-3.39	-1.54	-1.44	-1.45	-1.06	0.15	2.08	2.08	2.08	2.42
40_1	-0.20	-0.30	-1.54	0.00	-0.01	-0.06	-0.15	0.00	0.00	0.00	0.00	0.00
40_2	-0.08	-1.09	-0.44	0.00	0.00	-0.06	-0.03	0.00	0.00	0.00	0.00	0.00
40_3	-5.18	-1.76	-0.49	0.00	0.52	0.91	0.93	1.58	4.16	4.95	4.16	4.95
50_1	-0.13	-0.32	-0.88	-0.12	-0.02	-0.06	-0.13	0.00	0.00	0.00	0.00	0.00
50_2	0.00	-0.55	-0.42	0.00	0.06	-0.02	0.00	0.06	0.29	0.29	0.29	0.29
50_3	-0.87	-1.03	-0.52	-0.30	1.11	0.88	1.05	1.52	2.85	2.85	2.85	2.85

Table 4.25: The performance of the best configuration in every single neighbourhood and the mixed neighbourhood (1).

Data	#Good (225)				#Better				
	EWC*	**Switch+**	**0.3-Swap+**	**Mix**	**EWC***	**Switch+**	**0.3-Swap+**	**Mix**	**Max**
30_1	25	19	17	25	0	0	0	0	0
30_2	24	22	22	25	0	0	0	0	0
30_3	4	2	5	17	1	1	1	5	10
40_1	22	12	7	25	0	0	0	0	0
40_2	23	18	20	25	0	0	0	0	0
40_3	17	14	18	25	14	13	13	20	20
50_1	20	16	13	24	0	0	0	0	0
50_2	25	15	17	25	10	7	6	15	25
50_3	19	18	21	23	19	18	21	23	25
Total	179	136	140	214	44	39	41	63	80

Table 4.26: The performance of the best configuration in every single neighbourhood and the mixed neighbourhood (2).

Data	Total time (s)				Time/Iteration (s)			
	EWC*	**Switch+**	**0.3-Swap+**	**Mix**	**EWC***	**Switch+**	**0.3-Swap+**	**Mix**
30_1	0.88	1.2	0	13.16	0.02	0.00	0.00	0.01
30_2	1.04	1.28	0.44	16.88	0.01	0.00	0.00	0.02
30_3	14.56	7.92	7.44	118.16	0.19	0.02	0.03	0.12
40_1	2.72	3.4	1.28	24.72	0.04	0.00	0.00	0.02
40_2	2.4	3.16	1.96	20.92	0.03	0.01	0.01	0.02
40_3	12.8	9.12	7.92	76.12	0.15	0.01	0.02	0.08
50_1	1.44	6.52	2.56	18.56	0.03	0.01	0.01	0.02
50_2	2.56	5.76	3.28	22.64	0.03	0.01	0.01	0.02
50_3	12.36	11	10.68	60.4	0.09	0.01	0.02	0.06

Table 4.27: The performance of the best configuration in every single neighbourhood and the mixed neighbourhood (3).

Data	EWC*	Switch+	0.3-Swap+	Mix	Max
30_1	11	9	8	11	14
30_2	10	12	11	10	13
30_3	21	17	16	24	25
40_1	18	20	19	18	24
40_2	19	14	16	14	19
40_3	23	22	22	22	24
50_1	13	17	17	7	21
50_2	22	19	20	19	25
50_3	23	21	23	22	25
Total	160	151	152	147	190

Table 4.28: The effectiveness of diversification in each neighbourhood.

Data	%Gap*	Best %Gap*	#Better	
			Mix	Max
30_1	1.04	5.27	14	14
30_2	1.03	5.19	13	13
30_3	4.71	14.09	24	25
40_1	1.20	6.84	24	24
40_2	0.81	2.62	19	19
40_3	4.36	13.04	24	24
50_1	2.00	9.68	21	21
50_2	1.11	2.69	24	25
50_3	3.48	8.89	24	25
Total			187	190

Table 4.29: The performance of the tabu search with the mixed neighbourhood to improve the initial solution.

number of better solutions in Columns *#Better* where *Mix* and *Max* show, respectively, the number of better solutions from the mixed neighbourhood and the maximum possible cases. Note that the information in *Max* is from the number of results out of 25 whose initial solution is not the optimal solution, which is the same as in Column *Max* of Table 4.28. We do not focus on *Worst %Gap** and *#Good* here because we accept only a better solution than an initial solution in the tabu search.

Clearly, the tabu search significantly improves the quality of the initial solutions: see positive values in *% Gap** and *Best %Gap**. More importantly, the total of *#Better* supports that it finds a better solution in almost every possible case.

4.6.9 Comparison between the Tabu Search and Benchmark

We compare the performance of our best tabu search algorithm, i.e, the mixed neighbourhood or *Mix*, to the benchmark in Table 4.30. We present Columns *Worst %Gap**, *%Gap**, *Best %Gap**, *#Good (225)*, and *#Better* as usual. Also, we show the additional information of the number of results that find the optimal solutions in Columns *#Opt* where Sub-column *Max* suggests the maximum possible cases (which is the same as in Sub-column *Max* of Columns *#Opt* in Table 4.3). In terms of average time, Columns *Total time (s)* show the average computational time of the results in the mixed neighbourhood and the benchmark in Sub-columns *Mix* and *BM*, respectively.

In the combinations that find the optimal solution in every data instance, i.e., 30_1, 30_2, 40_1, 40_2, and 50_1, we aim to find at least as good solutions as these optimal solutions in every result. A number of *#Good* and *#Opt* by the mixed neighbourhood in these combinations show that *Mix* is effective in finding those optimal solutions in every result, except only for one result in 50_1.

When the data instances become more challenging to solve, such as in 30_3, 40_3, 50_2, and 50_3, the exact solution approaches struggle to find the optimal solutions as shown in Sub-column *Max* of Columns *#Opt*. In these combinations, except for 30_3, we can find comparable solutions to the benchmark for almost every result. In 30_3, *Mix* finds comparable solutions and can find the optimal solutions in 68% and only 47% of the possible cases, respectively: see the corresponding number of *#Good* and *#Opt*. However, overall *Mix* still has a high total of *#Good* and *#Opt*, which reach, respectively, at least 95% and 93% of the possible total number of cases.

In terms of *#Better*, for the more challenging combinations, *Mix* is capable of finding better solutions than the benchmark: note the positive values in *%Gap** and *Best %Gap** for the corresponding cases. In particular, *Mix* finds better solutions in 50% and 60% of the possible better cases in the combinations 30_3 and 50_2, respectively. For the most complicated combinations, such as 40_3 and 50_3, it finds better solutions in more than 95% of the possible

better cases. As a result, the total of *#Better* is 79% of the total number of possible cases, which is considerably high.

Regarding the average computational time, *Mix* shows outstanding performances especially when the data instances become more challenging. In fact, *Mix* spends at least 80% less time than the benchmark in those instances, which confirms the robustness of the algorithm.

In conclusion, *Mix* is capable of finding good-quality solutions and even better solutions within significantly less amount of average time, especially in the more challenging combinations. Therefore, *Mix* outperforms the benchmark.

4.6.10 Comparison between the CPLEX's Heuristics and the Tabu Search

According to *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual* (2017), CPLEX has additional built-in heuristics to enhance the efficiency of solving difficult mixed-integer problems. Namely, it provides heuristics to find integer solutions at nodes (including the root node) during the branch-and-cut process. In the default setting, CPLEX usually decides automatically if it will use the heuristics during the solving process. Users then can control how to use the heuristics by modifying the values of a set of corresponding parameters.

Here, we focus on feasibility pump, Relaxation Induced Neighbourhood Search (RINS), and solution polishing. The feasibility pump aims for finding an initial feasible solution at the beginning so it is often activated before the branch-and-cut procedure. RINS and the solution polishing are used to find more good feasible solutions. RINS exploits the information of a fractional solution at a current node and a current incumbent to create a neighbourhood of the current incumbent. In particular, the neighbourhood is formulated as another mixed-integer problem and searched for an improving incumbent. Therefore, in the default setting, CPLEX activates RINS less frequently than any of its regular heuristics at a node since it requires more computational time. The solution polishing is different from RINS as it is a branch-and-cut procedure itself. It is suitable where good solutions are hard to find. However, it is much more expensive in terms of computational time than other heuristics in CPLEX. Therefore, it is not called at all during the branch-and-cut process. Instead, it is usually used in a second phase to further improve the current best-known solution. More specifically, it requires an initial solution, e.g., from the original branch-and-cut procedure or a user-defined solution, for polishing. Moreover, it solely aims for improving the given solution so it does not guarantee the optimality condition.

In this section, we compare solutions from the feasibility pump to our initial solutions from the location-allocation heuristic. Then, we compare the performance of RINS and the solution polishing to the tabu search algorithm, where they all have the same initial solution from the location-allocation heuristic to begin the procedures. This is to get a fair comparison of the efficiency of improving the initial solutions.

Comparison between the Feasibility Pump and the Location-Allocation Heuristic

There are two options to control the quality of solutions from the feasibility pump. The first option is only to get a feasible solution but does not seek for solution quality, while the second option takes the objective value into account. This means that the second option seeks solutions with better objective values. However, it tends to fail in finding those solutions.

We set the maximum time for the feasibility pump at one minute. Our preliminary results support the above discussion that the second option cannot find any feasible solution in some data instances, while the first option can find a solution in every case. Therefore, we choose the first option of the feasibility pump and compare its performance to the location-allocation heuristic.

Table 4.31 shows the comparison of the feasibility pump to the location-allocation heuristic in terms of the solution quality and average time in seconds. Namely, it shows the usual information, e.g., *Worst %Gap**, *%Gap**, *Best %Gap**, *#Good (225)*, and *#Better*. Note that in Columns *#Better*, the results from the feasibility pump and the maximum possible cases in each combination are presented in Sub-columns *FP* and *Max*, respectively. *Max* here is the

Data	Worst % Gap*	% Gap*	Best %Gap*	#Good (225)	#Opt		#Better		Total time (s)	
					Mix	Max	Mix	Max	Mix	BM
30_1	0.00	0.00	0.00	25	25	25	0	0	13.16	6.4
30_2	0.00	0.00	0.00	25	25	25	0	0	16.88	120
30_3	-1.54	0.15	2.42	17	7	15	5	10	118.16	586.2
40_1	0.00	0.00	0.00	25	25	25	0	0	24.72	29.6
40_2	0.00	0.00	0.00	25	25	25	0	0	20.92	291.2
40_3	0.00	1.58	4.95	25	5	5	20	20	76.12	820.6
50_1	-0.12	0.00	0.00	24	24	25	0	0	18.56	44.2
50_2	0.00	0.06	0.29	25	0	0	15	25	22.64	900.8
50_3	-0.30	1.52	2.85	23	0	0	23	25	60.4	902.4
Total				214	136	145	63	80		

Table 4.30: The comparison of the performance between the best-found solutions in Chapter 3 and the mixed neighbourhood.

Data	Worst %Gap*	%Gap*	Best %Gap*	#Good (225)	#Better		Total time (s)
					FP	Max	
30_1	-57.14	-26.35	-8.04	0	0	14	0.54
30_2	-199.95	-57.45	-4.24	0	0	13	1.57
30_3	-248.49	-146.68	-103.07	0	0	25	3.94
40_1	-55.64	-25.02	-6.73	0	0	24	1.44
40_2	-216.04	-61.73	-17.80	0	0	19	6.04
40_3	-171.79	-107.87	-25.37	0	0	24	13.34
50_1	-81.12	-39.54	3.91	1	1	21	3.46
50_2	-71.00	-40.34	-20.12	0	0	25	13.14
50_3	-376.77	-226.85	-151.31	0	0	25	45.49
Total				1	1	190	

Table 4.31: The performance of the feasibility pump compared to the location-allocation heuristic.

same as in Sub-column *Max* of Columns *#Better* in Table 4.29, where the initial solutions are not the same optimal solutions found by the benchmark from Section 4.6.1.

Clearly, the overall solution quality from the feasibility pump is considerably worse than that from the location-allocation heuristic: see the high negative relative percentage gaps even in *Best %Gap**. In fact, for the most complicated combinations, such as 30_3, 40_3, and 50_3, the average relative percentage gaps are between 100% and 230% worse. Also, the feasibility pump can find only one better result out of 190 possible cases in total.

In terms of average computational time, the feasibility pump tends to spend much more computational time when the data instances become more complicated, while the location-allocation heuristic requires less than one second to derive an initial solution.

Due to higher solution quality and a significantly less amount of computational time, the location-allocation heuristic is incredibly more effective and efficient than the feasibility pump to derive good initial solutions.

Comparison between RINS, the Solution Polishing, and the Tabu Search

To have a fair comparison, we provide the same initial solutions from the tabu search to RINS and the solution polishing at the beginning. The maximum time to run the experiments is 900 seconds, similarly to the tabu search with the single neighbourhoods. Also, we control the maximum number of times to improve a solution in CPLEX's heuristics. Namely, we limit the number of times to at most 1,000 which is the same stopping criterion of the tabu search with the mixed neighbourhood. For RINS, we can update the frequency in terms of node interval to apply the heuristic. Since it may be expensive to apply it in every node, we choose to activate it in every 20 nodes. We set up the maximum number of nodes to 20,000 nodes so that it has a chance to improve a solution for at most 1,000 times. For the solution polishing, it is applied in every node and we cannot change this setting. Therefore, we restrict the number of nodes to at most 1,000 instead.

The performances of RINS and the solution polishing compared to the tabu search with the mixed neighbourhood are, respectively, represented by *RINS* and *solPol* in Tables 4.32 and 4.33. We present the same information as before, i.e, *Worst %Gap**, *%Gap**, *Best %Gap**, *#Good (225)*, *#Better*, and *Total time (s)*. In Columns *#Better*, the information in Sub-column *Max* for each combination is from the number of results out of 25 where the best solutions from the tabu search are not the same optimal solutions found by the benchmark from Section 4.6.1; it is implied from the information in Sub-column *Mix* of Columns *#Opt* in Table 4.30. Namely, these cases still have the opportunity to get better solutions. In Columns *Total time (s)*, we present the information of the average computational time of the CPLEX heuristics. Also, we show the average time in the tabu search with the mixed neighbourhood as additional information in Sub-column *Mix*.

Data	Worst %Gap*		%Gap*		Best %Gap*	
	RINS	**solPol**	**RINS**	**solPol**	**RINS**	**solPol**
30_1	0.00	0.00	0.00	0.00	0.00	0.00
30_2	0.00	-0.42	0.00	-0.02	0.00	0.00
30_3	-7.78	-7.78	-0.93	-2.52	1.35	0.29
40_1	0.00	0.00	0.00	0.00	0.00	0.00
40_2	0.00	-0.51	0.00	-0.07	0.00	0.00
40_3	-8.29	-10.03	-2.96	-2.95	0.00	0.00
50_1	0.00	0.00	0.00	0.00	0.12	0.12
50_2	-0.81	-0.11	-0.09	-0.03	0.00	0.00
50_3	-9.21	-6.27	-2.63	-2.14	0.00	0.00

Table 4.32: The performance of RINS and the solution polishing compared to the tabu search (1).

Data	#Good (225)		#Better			Total time (s)		
	RINS	**solPol**	**RINS**	**solPol**	**Max**	**RINS**	**solPol**	**Mix**
30_1	25	25	0	0	0	48.99	686.16	13.16
30_2	25	24	0	0	0	131.74	900.02	16.88
30_3	18	7	8	1	18	900.02	900.02	118.16
40_1	25	25	0	0	0	85.29	900.02	24.72
40_2	25	20	0	0	0	447.50	900.04	20.92
40_3	2	3	0	0	20	900.02	900.05	76.12
50_1	25	25	1	1	1	148.44	900.05	18.56
50_2	15	15	0	0	25	900.02	900.06	22.64
50_3	2	2	0	0	25	900.02	900.04	60.4
Total	162	146	9	2	89			

Table 4.33: The performance of RINS and the solution polishing compared to the tabu search (2).

First, we compare the performance between RINS and the solution polishing. Table 4.32 shows that there is no significant difference in terms of the relative percentage gaps. Table 4.33 provides more insights about the quality of solutions. *RINS* is overall superior to *solPol* in terms of finding comparable and better solutions: see the higher total of *#Good* and that of *#Better*. When both heuristics do not reach the maximum time, *RINS* spends a significantly less amount of time on average to derive those solutions. Therefore, in this case, we conclude that RINS has better performance than the solution polishing.

Now, we investigate the performance of RINS compared to the tabu search. The number of *#Good* in Table 4.33 supports that it is comparable in less complicated combinations, such as 30_1, 30_2, 40_1, 40_2, and 50_1. However, in more challenging combinations the relative percentage gaps in Table 4.32 tend to be negative which show that RINS struggles to find even comparable solutions. As a result, it fails to derive better solutions in many data instances. In particular, it finds better solutions only in 10% of the total number of possible cases: see the total number of *#Better*. More importantly, RINS spends incredibly more time on average as shown in Columns *Total time (s)*, especially when the data instances become more complicated.

Therefore, from the performance in terms of the solution quality and average time, the CPLEX's heuristics have significantly worse performances than the tabu search.

4.6.11 Improvement on the Benders' Decomposition by the Tabu Search

In the previous sections, we showed the robustness of the tabu search with the mixed neighbourhood. Now, we use the tabu search to improve the efficiency of the Benders' decomposition algorithm from the previous chapter.

In the beginning, the tabu search is used to improve the quality of an initial solution for the Benders' decomposition method. After that, it is activated again whenever the Benders' decomposition algorithm finds a new incumbent. If the tabu search is successful in the solution improvement, we will provide the improved solution to the Benders' decomposition algorithm via the *heuristic callback*. Note that this callback is for injecting any feasible solution during the branch-and-cut process (*IBM ILOG CPLEX Optimization Studio CPLEX User's Manual* 2017). According to IBM (2021) the built-in Benders' algorithm in CPLEX, however, does not support additional features from callbacks written by a user. Therefore, we combine the tabu search in our own Benders' decomposition instead and investigate its improvement.

We set up for our Benders' algorithm with the same setting in Chapter 3. Also, we apply the same tabu search with the mixed neighbourhood in the solution improvement process. Namely, the stopping criterion every time that we activate the tabu search is 1000 iterations. Also, we keep the information of the probabilities to select a neighbourhood and reinitialise the tabu lists for the next time that we apply the tabu search. Finally, the maximum time of the Benders' decomposition algorithm including the computational time to run the tabu search is still 900 seconds.

Comparison with our Previously Developed Benders' Decomposition Algorithm

In this section, we investigate if the tabu search can enhance the efficiency of the previously developed Benders' decomposition algorithm. We call the algorithm that combines Benders' decomposition and the tabu search *combined algorithm*. The benchmark here is our Benders' decomposition from the previous chapter.

Table 4.34 shows the information on the efficiency of the tabu search in the algorithm. First, Columns *%DiffLB* present the relative percentage gap between the lower bound of the objective function value from the combined algorithm and the benchmark. Sub-columns *Worst*, *Average*, *Best* are, respectively, the worst, the average, and the best relative percentage gap in the aspect. A similar comparison between the best integer value in both algorithms is in Columns *%DiffUB*. Note that the positive sign of the relative percentage gap implies better performance, while the negative sign is the opposite. In particular, better performance means a smaller best integer value or a higher lower bound. Also, we present the regular information of the number of results that are comparable and better than the benchmark in Columns *#Good (45)* and *#Better*, respectively. The maximum total number of possible cases for *#Good* is the total number of data instances which is shown in the brackets. In Columns *#Better*, Sub-column *PapaTS* presents the number of results from the combined algorithm that have better solutions than the benchmark, while Sub-column *Max* shows the corresponding limit. In particular, the information of *Max* is the number of data instances that the benchmark have not found the optimal solutions yet, which is implied from Sub-column *Papa* of Columns *#Opt (45)* in Table 3.10.

Columns *%DiffLB* and *%DiffUB* show that there is no significant difference in terms of the lower bound and the best integer solution in the less complicated instances, e.g., 30_1, 40_1, and 50_1. When the data instances become more complicated like in the combinations 30_3, 40_3, and 50_3, the best integer solution from the combined algorithm is better in almost every case: see the corresponding non-negative values of *Worst %Gap** and the values of *#Better*. Moreover, the total of *#Good* and that of *#Better* support the high-quality of the best integer solutions from the combined algorithm. In fact, the combined algorithm finds comparable solutions and better solutions in 98% and 73% of total number of possible cases. However, most values in Columns *%DiffLB* for the more complicated instances are negative, implying that the lower bound in these cases tends to be slightly worse than the benchmark.

From the above discussion, the combined algorithm gains the benefit of finding better integer solutions from the tabu search. However, it tends to have a worse lower bound of the objective

function than our previously developed Benders' decomposition algorithm.

Overall Comparison to the Benders' Decomposition Algorithms

Now, we compare the performance of the combined algorithm to the Benders' decomposition algorithms in the previous chapter in Tables 4.35 and 4.36, where *AutoBD*, *Papa*, and *PapaTS* represent the built-in Benders' decomposition algorithm in CPLEX, our Benders' decomposition without and with the tabu search, respectively. Here, we focus on the relative percentage gap of the objective value, the number of results that can find the optimal solutions, and the average computational time.

Table 4.35 shows the information of the relative percentage gap between the upper bound and the lower bound of the objective value by the algorithms. Namely, in each combination, the best, the average and the worst relative percentage gap among 5 data instances are presented in Columns *Best %Gap*, *%Gap*, and *Worst %Gap*, respectively. As discussed before, although the combined algorithm is capable of finding better integer solutions than the Benders' algorithms from the previous chapter, its overall percentage gaps here are not outstanding due to lower values in the lower bounds. Also, the combined algorithm has the least total number of results that find the optimal solutions: see the corresponding total number of *#Opt* in Table 4.36.

Columns *Total time (s)* of Table 4.36 present the information of each algorithm in terms of the average computational time. We focus on the less complicated data instances in which all algorithms complete their process before the maximum time, e.g., in the combinations 30_1, 30_2, 40_1, and 50_1. In particular, it is possible to find optimal solutions in these data instances. Clearly, the combined algorithm spends significantly more time to complete the process due to the additional time to run the tabu search.

As shown before in Section 4.6.6, the tabu search requires some amount of times to reach its stopping criterion, especially in more complicated data instances. As a result, the combined algorithm may have less time to improve the lower bound during the branch-and-cut process, thereby achieving a worse value of the lower bound. In such cases, the improvement in the best integer solution may not compensate for those deteriorating performances.

Summing up, the tabu search is capable of finding better integer solutions. However, the combined algorithm tends to result in a worse lower bound of the objective value. Therefore, a more elaborate design to combine the tabu search with Benders' decomposition is still required such that the algorithm can exploit the advantages of the tabu search and control the quality of the lower bound at the same time.

4.6.12 Conclusion

In conclusion, we introduced three neighbourhoods in the tabu search algorithm: the week centre neighbourhood, the switching week pattern neighbourhood, and the swapping week pattern neighbourhood. We proposed simple but effective techniques to speed up the search in each neighbourhood. Then, we unified the best configuration of each neighbourhood into the mixed neighbourhood and compared its performances to the exact solution approaches from Chapter 3. The results showed that it is effective in finding high-quality solutions while spending much less time, especially when data instances become more challenging to solve. Moreover, we investigated the improvement of the initial solutions and the quality of the diversification to prove the effectiveness of the algorithm. Furthermore, we showed that the initial solutions and the best solutions from the tabu search algorithm compared favourably with the heuristics in CPLEX. Finally, we combined the tabu search with our previously developed Benders' decomposition. The advantages of the tabu search, however, cannot significantly improve the efficiency of the combined method so more proper development is still required in such case.

Due to the robustness of the tabu search, we will extend the method to solve the districting and the scheduling part of MPSDP simultaneously, which will be presented in the next chapter.

Data	%DiffLB			%DiffUB			#Good (45)	#Better	
	Worst	Average	Best	Worst	Average	Best		PapaTS	Max
30_1	-0.01	0.00	0.00	0.00	0.00	0.00	5	0	0
30_2	-0.86	-0.17	0.00	0.00	0.00	0.00	5	0	0
30_3	-6.96	-1.63	0.03	-0.71	1.11	6.03	4	3	4
40_1	0.00	0.00	0.00	0.00	0.00	0.00	5	0	0
40_2	-0.33	-0.14	0.00	0.00	0.00	0.00	5	0	3
40_3	-6.47	-1.22	3.46	0.14	2.29	4.95	5	5	5
50_1	0.00	0.00	0.00	0.00	0.00	0.00	5	0	0
50_2	-0.43	-0.25	0.02	0.00	0.13	0.55	5	3	5
50_3	-2.88	-0.40	3.93	0.29	1.86	2.85	5	5	5
Total							44	16	22

Table 4.34: The performances of the Benders' decomposition algorithm with the tabu search compared to our previously developed Benders' decomposition.

Data	Best %Gap			%Gap			Worse %Gap		
	AutoBD	**Papa**	**PapaTS**	**AutoBD**	**Papa**	**PapaTS**	**AutoBD**	**Papa**	**PapaTS**
30_1	0.00	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01
30_2	0.01	0.01	0.01	0.46	0.01	0.18	2.25	0.01	0.87
30_3	0.01	0.01	0.01	2.34	4.92	5.41	8.29	15.43	16.27
40_1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
40_2	0.01	0.01	0.01	0.15	0.49	0.63	0.70	0.95	1.28
40_3	0.01	4.12	5.53	12.40	14.21	13.26	21.29	19.09	22.39
50_1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
50_2	1.31	1.04	1.03	2.41	2.73	2.84	4.53	4.89	5.13
50_3	11.08	13.48	11.39	17.46	17.60	16.39	21.70	21.10	21.09

Table 4.35: The overall performance of different Benders' decomposition algorithms (1).

Data	#Opt (45)			Total time (s)		
	AutoBD	**Papa**	**PapaTS**	**AutoBD**	**Papa**	**PapaTS**
30_1	5	5	5	9	7.6	38.2
30_2	4	5	4	290.4	261	353.8
30_3	3	1	1	624	777.8	813.4
40_1	5	5	5	48.6	102	266.8
40_2	4	2	2	677.4	727.2	734.4
40_3	1	0	0	819	904	900
50_1	5	5	5	57.8	66.4	143.8
50_2	0	0	0	900	901.8	901
50_3	0	0	0	900	902.6	900.2
Total	27	23	22			

Table 4.36: The overall performance of different Benders' decomposition algorithms (2).

Chapter 5

Extended Tabu Search for the MPSDP

According to Bender et al. (2016), the MPSDP contains two parts of decision planning: the districting part and the scheduling part. The districting part is a strategic plan to assign customers to each salesman which is usually run long-term, e.g., at least for several years. In contrast, the scheduling part, which aims to create effective weekly schedules for salesmen to visit customers, is a tactical plan and requires revision more frequently, e.g., every few months. The study of districting problems has been extensive already, but the study of scheduling usually did not take into account geographical compactness prior to Bender et al. (2016). Therefore, Bender et al. (2016) focused on implementing an effective solution method only for the scheduling part. Whenever the districting part needs to be revised, they suggested solving the parts sequentially: first solving the districting part as in the classical sales districting problem and then solving the scheduling part. However, following their suggestion, we might end up with a suboptimal solution for the MPSDP. Therefore, it is preferable to develop a solution approach that solves the districting part and the scheduling part at the same time.

As the tabu search algorithm is successful in solving the scheduling part, it is desirable to extend the method to include the districting part, i.e., to solve the whole MPSDP. The extension of the tabu search algorithm is described in Section 5.1. Section 5.2 presents the information on setting up the experiment, followed by the computational results to show the robustness of the algorithm in Section 5.3.

5.1 Extension of the Tabu Search Algorithm

In Chapter 4, we presented the tabu search algorithm for solving the scheduling part of the MPSDP, and the computational results showed the effectiveness of the algorithm. Here, we extend the method to solve the districting and the scheduling part simultaneously.

We assume that we start with a set of salesmen's offices and a set of customers, including information on the customers' requirements regarding visiting frequency and amount of service time. The tasks for the MPSDP include partitioning the customers into districts for the salesmen (the districting part) and, at the same time, allocating the customers to weekly schedules (the scheduling part). Both parts aim for the same favourable attributes: geographical compactness and workload balance. For the districting part, the measurement of the compactness is the centre-based distance between a salesman's office and customers allocated to the office. At the same time, the workload in a district, which is calculated from the total workload to serve the assigned customers in the whole planning horizon, has to be within an acceptable range. The main reason for this is that the workload balance on the district level indicates fairness among salesmen, so it is a top priority. In the case of the scheduling part, the compactness and the workload in each weekly schedule are measured in a similar way as in Chapters 3 and 4. Also, we allow the weekly schedules to be infeasible in terms of weekly workload balance during the search to increase the search space. To prevent the search from focusing on those infeasible solutions too much, we penalise them by a self-adjusting penalty factor.

We formulate a solution for the MPSDP that contains the necessary information for every district. For each district, we record the set of allocated customers and the information on weekly schedules, i.e., a vector of customers who are week centres and a vector of week patterns of every customer.

Let $D = \{1, \ldots, |D|\}$ be the set of districts. A solution of the MPSDP is represented by $\mathfrak{D} \equiv (\mathfrak{D}^1, \mathfrak{D}^2, \ldots, \mathfrak{D}^{|D|})$, where $\mathfrak{D}^i$ is a solution for district $i \in D$ which is defined in the following way:

$\mathfrak{D}^i \equiv (\mathfrak{B}^i, \omega^i, \rho^i)$, where $\mathfrak{B}^i \subseteq B$ is the set of allocated customers in district i. Similarly to a solution for the scheduling part in the previous chapters, ω^i and ρ^i are a vector of customers who are week centres and a vector of week patterns of customers, respectively, i.e.,

- $\omega^i \equiv (\omega^i_1, \omega^i_2, \ldots, \omega^i_{|W|})$, where ω^i_w is the customer who is the week centre of district i in week w, for $w = 1, 2, \ldots, |W|$. Note that any week centre is not required to be in the district because it is only for measuring the compactness of the weekly schedules.
- $\rho^i \equiv (\rho^i_1, \rho^i_2, \ldots, \rho^i_{|\mathfrak{B}^i|})$, where ρ^i_b is the week pattern of customer $b \in \mathfrak{B}^i$ in district i. The corresponding information of ρ^i is $\delta(\rho^i)$ to evaluate the total excess of weekly workloads from an acceptable range during the planning horizon.

The objective value of $\mathfrak{D}^i$ contains two parts of the compactness measurement: the total centre-based distance of the district and the total centre-based distance arising from the weekly schedules. Let $\bar{c}_{bi}$ be the distance between customer $b \in \mathfrak{B}^i$ and the centre of district i, and c_{bn} be the distance between customer b and n, for $b, n \in B$. For the scheduling part, we define $\mathfrak{B}^i_w$ as the set of customers of district i to be served in week w. It is straightforward to derive this set from the information of ρ^i, as shown in Chapter 4. Also, we have ϕ as the penalty factor to penalise infeasible weekly schedules.

Let $ComDis(\mathfrak{D}^i)$ and $ComSch(\mathfrak{D}^i)$ be the compactness on the district level and on the week level, which are calculated as follows:

$$ComDis(\mathfrak{D}^i) = \sum_{b \in \mathfrak{B}^i} \bar{c}_{bi} \tag{5.1}$$

$$ComSch(\mathfrak{D}^i) = \sum_{w \in W} \sum_{b \in \mathfrak{B}^i_w} c_{b,\omega_w} + \phi \cdot \delta(\rho^i). \tag{5.2}$$

The objective value of $\mathfrak{D}^i$ represented by $z(\mathfrak{D}^i)$ is calculated as

$$z(\mathfrak{D}^i) = \lambda \cdot ComDis(\mathfrak{D}^i) + (1 - \lambda) \cdot ComSch(\mathfrak{D}^i) \tag{5.3}$$

where λ is a weight of how much emphasis we put on the compactness on the district level.

Then, the objective value of $\mathfrak{D}$, $z(\mathfrak{D})$, is the sum of the objective values of every district, i.e.,

$$z(\mathfrak{D}) = \sum_{i \in D} z(\mathfrak{D}^i). \tag{5.4}$$

A feasible solution $\mathfrak{D}$ that we accept during the search has to pass two requirements. First, the assignment of customers to districts must be complete and exclusive, i.e., $\cup_{i \in D} \mathfrak{B}^i = B$ and $\mathfrak{B}^i \cap \mathfrak{B}^j = \emptyset$, for $i, j \in D, i \neq j$. This is to guarantee that a customer will be served by a unique salesman during the planning horizon. Moreover, we do not accept any solution that violates the workload balance of districts in the algorithm. In particular, every solution $\mathfrak{D}^i$, $i \in D$ has to satisfy Constraints (2.16) and (2.17), as follows:

$$(1 - \tau^{dis})\mu^{dis} \leq size(\mathfrak{D}^i) \leq (1 + \tau^{dis})\mu^{dis}$$

where $size(\mathfrak{D}^i) = \sum_{b \in \mathfrak{B}^i} \frac{t_b \cdot |W|}{r_b}$ is the size of district $i \in D$ which is the total workload during the planning horizon in the district. $\mu^{dis} = \sum_{b \in B} \frac{t_b |W|}{r_b |D|}$ is the average workload of districts, and τ^{dis} is a predefined maximum deviation from the average workload.

To derive an initial solution to start the algorithm, we first solve the classical sales districting problem to partition customers into districts. We adapt the part of the model formulation of the MPSDP in Section 2.4.2 of Chapter 2 that relates to partitioning customers. The adapted formulation is presented in the following (AllocationDis), where v_{bd} are the binary variables to

decide if customer $b \in B$ is allocated to district $d \in D$.

$$\min \quad \sum_{d\in D}\sum_{b\in B} \overline{c}_{bd} v_{bd} \qquad \text{(AllocationDis)}$$

$$\text{s.t.} \quad \sum_{d\in D} v_{bd} = 1 \qquad b \in B \qquad (5.5)$$

$$\sum_{b\in B} \frac{t_b \cdot |W| \cdot v_{bd}}{r_b} \geq (1-\tau^{dis})\mu^{dis} \qquad d \in D \qquad (5.6)$$

$$\sum_{b\in B} \frac{t_b \cdot |W| \cdot v_{bd}}{r_b} \leq (1+\tau^{dis})\mu^{dis} \qquad d \in D \qquad (5.7)$$

$$v_{bd} \in \{0,1\} \qquad b \in B,\ d \in D.$$

The objective function is to maximise the compactness on the district level. Constraints (5.5) guarantee the exclusive and complete assignment of customers. The workload balance of districts is ensured by Constraints (5.6)−(5.7), where t_b and r_b are the amount of required service time and the week rhythms of customer $b \in B$, respectively. Overall, this problem guarantees every customer is assigned to a unique salesman, while the workload in each district is within an acceptable range.

We derive the set $\mathfrak{B}^i$ for each district $i \in D$ from the values of variable v_{bd}, for $b \in B, d \in D$, as follows: $b \in \mathfrak{B}^i$, if $v_{bi} = 1$, for $b \in B,\ i \in D$.

Next, in district i with set $\mathfrak{B}^i$, we derive the information of weekly schedules, ω^i and ρ^i, in the same way as an initial solution for the scheduling part in Chapter 4. We select an initial week centre each week randomly among specific customers in the district. Those customers are the ones who have the smallest week rhythm in the planning horizon. Then, we run the allocation-location heuristic to improve the quality of weekly schedules. Note that now any customer in B can be a week centre without having to be in the district. We prefer feasible weekly schedules to start the algorithm. In the case where we get infeasible weekly schedules from the location-allocation heuristic, we start with a new vector of random initial week centres and repeat the process. We allow the repetitions for at most a fixed number of times.

In the algorithm, we run a separate tabu search to improve the districting part and the scheduling part of a solution iteratively. In particular, we run the tabu search algorithm for the districting part to exchange customers between their adjacent districts. Then, in each district, we run the tabu search for the scheduling part to improve the quality of weekly schedules corresponding to a new set of allocated customers. Chapter 4 has already presented the tabu search algorithm to improve the scheduling part: the tabu search algorithm with the mixed neighbourhood. Here, we will describe the tabu search for the districting part and how to integrate these two parts in the main algorithm.

5.1.1 Tabu Search for the Districting Part

Regarding the tabu search algorithm for the districting part, we have two simple operations: moving a customer to an adjacent district and swapping two customers whose districts are adjacent. To search for an adjacent district(s) for each customer $b \in B$, we investigate the customers who are closest to customer b and check their districts. If those customers belong to different districts from that of b, those districts are deemed neighbouring and become options for a new district for b.

Namely, for customer b, we create a list L^b to sort all customers $b' \in B \backslash \{b\}$ in non-decreasing order of $c_{b,b'}$ (the distance between b and b') so that customers who are closest to b are the first elements. To focus only on customers who surround b, we limit the number of customers to investigate in L^b corresponding to a parameter γ, $0 < \gamma < 1$. Let $N_\gamma = \gamma \cdot |B|$ be the reduced number of customers to investigate in list L^b. In particular, the value of γ should be small enough to investigate just the customers who are closest to b. We consider the first N_γ customers in L^b and check their current districts. If any of them is in a different district from the current district of b, we record those customers in a set $\hat{B}^b_\gamma$. If the set is not empty, there is a candidate(s) for a new district of customer b in the search.

Assume that we have a current solution $\mathfrak{D}$. We define $Move(b, i, k)$ as a move operation for moving any customer b from the current district i to a new adjacent district k. After we apply the move operation, we update the information of district i and k that gets changed: their size and their new objective values from the districting part and the scheduling part.

The size of districts that are affected by the move operation are updated in the following way,

$$size(\mathfrak{D}^i) = size(\mathfrak{D}^i) - \frac{t_b \cdot |W|}{r_b} \tag{5.8}$$

$$size(\mathfrak{D}^k) = size(\mathfrak{D}^k) + \frac{t_b \cdot |W|}{r_b}. \tag{5.9}$$

Then, we update the objective values of the changed districts. A way to update the objective value of the districting part and the scheduling part for those districts is simple: we delete any related information of customer b in the old district and add it to the new district. The updated objective value of the districting part of each district is represented below.

$$ComDis(\mathfrak{D}^i) = ComDis(\mathfrak{D}^i) - c_{bi} \tag{5.10}$$

$$ComDis(\mathfrak{D}^k) = ComDis(\mathfrak{D}^k) + c_{bi}. \tag{5.11}$$

Regarding the updated objective value of the scheduling part, in tabu search for the districting part, we do not focus on improving any weekly schedule. Therefore, we approximate the objective value of the scheduling part. We keep the week centres of the old district and the new district and update only the week pattern of customer b in the new district. The new week pattern for customer b in district k, ρ_b^{k*}, is the one that has the least total centre-based distance between b and the corresponding week centre vector ω^k, i.e.,

$$\rho_b^{k*} = \arg\min_{p \in P_b} \sum_{w \in W} \psi_p^w c_{b,\omega_w^k}. \tag{5.12}$$

Then, the updated (approximate) compactness of the scheduling part of the old district and the new district, represented by $\overline{ComSch(\mathfrak{D}^i)}$ and $\overline{ComSch(\mathfrak{D}^k)}$, are calculated as

$$\overline{ComSch(\mathfrak{D}^i)} = \overline{ComSch(\mathfrak{D}^i)} - \sum_{w \in W} \psi_{\rho_b^i}^w c_{b,\omega_w^i} \tag{5.13}$$

$$\overline{ComSch(\mathfrak{D}^k)} = \overline{ComSch(\mathfrak{D}^k)} + \sum_{w \in W} \psi_{\rho_b^{k*}}^w c_{b,\omega_w^k} \tag{5.14}$$

where ρ_b^i and ρ_b^{k*} are the week patterns of b in the old district and the one from Equation (5.12) in the new district, respectively.

Now, we have the updated objective value in both districting and scheduling part for the old district and the new district, resulting in the new objective value for a neighbouring solution after one move operation. Note that we only accept a solution if every district does not violate the workload balance of the districts, i.e., every district of the solution satisfies Constraints (2.16) and (2.17).

The above steps related to the move operation $Move(b, i, k)$ on a solution $\mathfrak{D}$ are presented in Algorithm 15 (Move$(\mathfrak{D}, b, i, k)$).

For the case of swapping customers between two adjacent districts, we apply the move operation twice. For example, to swap b_1 and b_2 whose current districts are dis_{b_1} and dis_{b_2}, respectively, in solution $\mathfrak{D}$, we apply Move$(\mathfrak{D}, b_1, dis_{b_1}, dis_{b_2})$ and then apply Move$(\mathfrak{D}, b_2, dis_{b_2}, dis_{b_1})$. The updated solution $\mathfrak{D}$ after applying these algorithms is a neighbouring solution after swapping two customers between their districts.

To create the districting neighbourhood, we use the move operation to move one customer to an adjacent district or swap two customers whose districts are adjacent. After finishing the search in the neighbourhood, we store the best neighbouring solution and record the tabu(s) on the tabu list. The tabu here is a customer who has changed their district and their old district. For example, if we move customer b from the current district dis_b to a new one, the tabu is

Algorithm 15 The Operation to Move Customer b From District i to District j on a solution $\mathfrak{D}$ (Move($\mathfrak{D}, b, i, k$))

Input: $\mathfrak{D}, b, i, k$

1: $\mathfrak{B}^i = \mathfrak{B}^i \setminus \{b\}$ ▷ Delete the customer from the old district

2: $\mathfrak{B}^k = \mathfrak{B}^k \cup \{b\}$ ▷ Add the customer to the new district

3: Update $size(\mathfrak{D}^i)$ and $size(\mathfrak{D}^k)$ by Equation (5.8) and (5.9), respectively

4: Update the objective value of $\mathfrak{D}^i$ in the districting part and the (approximate) scheduling part by Equation (5.10) and (5.13)

5: Update the objective value of $\mathfrak{D}^k$ in the districting part and the (approximate) scheduling part by Equation (5.11) and (5.14)

Output: $\mathfrak{D}$

(b, dis_b). For the case of swapping two customers, we change the allocation of those customers to their new districts, so we record two tabus in the list. The tabu list has a fixed length, so each tabu stays in the list for a fixed number of iterations.

Algorithm 16 shows the search in the districting neighbourhood. Assume that we have the current solution $\mathfrak{D}$ and the tabu list T_D. Let z^* be the best neighbouring solution value in the neighbourhood and $\mathfrak{D}'$ be a neighbouring solution. In the algorithm, we focus only on customers who have a new district(s) to move to, i.e., by investigating which districts their surrounding customers belong to. In the exhaustive search, we explore every customer who has options of a new district to move in set $\overline{B}$ where $\overline{B} = \{b \mid b \in B \text{ and } \hat{B}^b_\gamma \neq \emptyset\}$. Line 5 shows that we apply the move operation for moving one customer to their adjacent district if it does not cause a tabu violation. Then, we check if a neighbouring solution is feasible in terms of the workload balance of districts and better than the currently best neighbouring solution, as shown in Line 6. If it passes these conditions, we update z^*, and $\mathfrak{D}^*$ as the new best neighbouring solution. Also, we update a tabu from the currently best neighbouring solution in set L^* in Line 10. For the case of swapping customers, we repeat the process of moving another customer in Lines 12−19. Finally, an element(s) in set L^*, which is recorded as a tabu(s) from the best neighbouring solution, is used to update the tabu list; see Line 23.

Reduced-size Districting Neighbourhood

To improve the effectiveness of the search, we focus on customers who are furthest from their current district centres. We investigate the distance between customer $b \in \overline{B}$ and the centre of their current district dis_b or $\overline{c}_{b,dis_b}$. We create a list L^{FC} that contains the customers $b \in \overline{B}$ in non-increasing order with respect to $\overline{c}_{b,dis_b}$. Therefore, the first customers on the list are the furthest ones from their district centres who can move to other districts. Let $N_\epsilon = \epsilon \cdot |B|$ be the reduced number of customers corresponding to a parameter $\epsilon \in \mathbb{R},\ 0 < \epsilon < 1$. We record the first N_ϵ customers of list L^{FC} in a set $\overline{B}_\epsilon$. In Line 1 of Algorithm 16, we replace set $\overline{B}$ with set $\overline{B}_\epsilon$. This limits the search in the districting neighbourhood to the customers who should move to other districts. We call the reduced-size districting neighbourhood with the positive parameter ϵ the *ϵ-reduced-size search on the districting neighbourhood* (ϵ-D).

Framework of Tabu Search for Districting Part

In the tabu search for the districting neighbourhood, we perform at least one iteration of the search to guarantee a change in districts, and then repeat it for at most a fixed number of iterations or until there is no improvement in the current solution. As we move only a few customers per iteration, the set $\overline{B}_\epsilon$ is unlikely to change for the reduced search. Therefore, in that case, we update the set $\overline{B}_\epsilon$ only once before every time we start the tabu search for the districting part. When any of the stopping criteria is reached, the next step of the main algorithm is to run the tabu search for the scheduling part in each district. Note that during the tabu search algorithm for the districting part, the objective values of the scheduling part of changed districts are calculated approximately by Equations (5.13) and (5.14). Therefore, at the end of the tabu search for the districting part, we improve the quality of weekly schedules

Algorithm 16 Exhaustive Search in Districting Neighbourhood of solution $\mathfrak{D}$ (ED($\mathfrak{D}$))

Input: $\mathfrak{D}$, T_D
Initial:
$z^* := 1e10$
for all $b_1 \in \overline{B}$ **do**
  for all $b_2 \in \hat{B}_{\gamma}^{b_1}$ **do** ▷ Observe the set of surrounding customers
    $\mathfrak{D}' := \mathfrak{D}$
    if $(b_1, dis_{b_2}) \notin T_D$ **then**
      Apply Algorithm Move($\mathfrak{D}', b_1, dis_{b_1}, dis_{b_2}$)
      ▷ Shift a customer to an adjacent district
      if $\mathfrak{D}'$ does not violate the workload balance of districts **and** $z(\mathfrak{D}') < z^*$ **then**
        $z^* := z(\mathfrak{D}')$
        $\mathfrak{D}^* := \mathfrak{D}'$
        $L^* := \emptyset$
        $L^* \cup \{(b_1, dis_{b_1})\}$ ▷ Update the set of tabu
      end if
      if $(b_2, dis_{b_1}) \notin T_D$ **then**
        Apply Algorithm Move($\mathfrak{D}', b_2, dis_{b_2}, dis_{b_1}$)
        ▷ Repeat the move operation for swapping the customers
        if $\mathfrak{D}'$ does not violate the workload balance of districts **and** $z(\mathfrak{D}') < z^*$ **then**
          $z^* := z(\mathfrak{D}')$
          $\mathfrak{D}^* := \mathfrak{D}'$
          $L^* \cup \{(b_2, dis_{b_2})\}$
        end if
      end if
    end if
  end for
end for
Add every element in L^* to T_D ▷ Update the tabu list
Output: $\mathfrak{D}^*$

in each district by applying the location-allocation heuristic, before starting the tabu search to improve the weekly schedules in each district further.

The tabu search algorithm for the districting part is presented in Algorithm 17. The parameter N_{Dis} is the maximum number of iterations which is one of the stopping criteria. i is a counter of the total number of iterations. To ensure that we change customers in some districts, we perform at least one iteration of the search and update the best neighbouring solution in $\overline{\mathfrak{D}}$; see Line 2. Note that for the reduced search, we update set $\overline{B}_\epsilon$ only once at the beginning of the tabu search. The stopping criteria of the algorithm are either non-improvement on the current solution $\mathfrak{D}'$ or the maximum number of iterations. Line 13 shows how to update the weekly schedules of each district at the end of the algorithm. It is similar to the location-allocation heuristic in the previous chapter. Here, we start the heuristic by solving the allocation problem with the current week centres of a district.

Algorithm 17 Tabu Search for the Districting Part

Input: $\mathfrak{D}$, T_D
Parameter: N_{Dis}
Initial:
$i := 1$
1: Update set $\overline{B}_\epsilon$ in case of the reduced search
2: $\overline{\mathfrak{D}} = \text{ED}(\mathfrak{D})$ ▷ Perform at least one iteration of Algorithm 16
3: **while** $i < N_{Dis}$ **do**
4: $i := i + 1$
5: $\mathfrak{D}' = \text{ED}(\overline{\mathfrak{D}})$
6: **if** $z(\mathfrak{D}') < z(\overline{\mathfrak{D}})$ **then**
7: $\overline{\mathfrak{D}} = \mathfrak{D}'$ ▷ Update the current solution for the next iteration
8: **else**
9: **Break** ▷ Stop the algorithm due to non-improvement on the current solution
10: **end if**
11: **end while**
12: **for all** $j \in D$ **do**
13: Update ω^j and ρ^j of $\overline{\mathfrak{D}}^j$ by the location-allocation heuristic ▷ Update the weekly schedules of $\overline{\mathfrak{D}}^j$
14: **end for**
Output: $\overline{\mathfrak{D}}$

5.1.2 Integrated Tabu Search for the MPSDP

Now, we have the tabu search algorithm for the districting part. In the main tabu search algorithm that integrates the districting part and the scheduling part, we run the tabu search to improve each part separately. For the districting part, we stop the corresponding tabu search algorithm when we reach the maximum number of iterations, or there is no improvement in the current solution. For the scheduling part, the stopping criterion is the maximum number of iterations. Note that the number of iterations in tabu search for both parts are independent.

When we start the tabu search for the scheduling part after updating customers in districts, we reinitialise the tabu lists of the mixed neighbourhood since we will search for weekly schedules for a new set of customers. Also, we update the length of the tabu list of the week patterns as it is related to the number of customers who do not require weekly services, which might be changed in the new set of customers. However, we still keep the probabilities of selecting a neighbourhood in the mixed neighbourhood, as the majority of customers in a district are almost the same and we believe that favourable neighbourhoods to find good weekly schedules for them are unlikely to change.

We iteratively improve each part until we reach the stopping criterion for the main algorithm, which is also the maximum number of iterations. Later, to avoid confusion, we define a *sub-iteration* as an iteration in the tabu search for improving either the districting part or the scheduling part, while a *main iteration* is an iteration in the main algorithm.

The tabu search algorithm to solve the MPSDP is presented in Algorithm 18, where N_{Dis} and N_{Sch} are the maximum number of sub-iterations to run tabu search for the districting part and the scheduling part, respectively, while N_{Main} is the maximum number of main iterations in the main algorithm. The counter of the main iterations is represented by it. Assume that we have an initial solution $\mathfrak{D}^0$ that is derived by solving (AllocationDis). Since the weekly schedules in each district of the initial solution are simply derived by the location-allocation heuristic, we improve them further by tabu search for the scheduling part; see Line 2. Then, we update the initial solution as the best neighbouring solution $\mathfrak{D}^*$. This process is counted as one main iteration as the solution has been updated by the tabu search for the scheduling part. In each main iteration, we apply tabu search for the districting part and then scheduling part, and update the better neighbouring solution, if we find one, as shown in Lines 6−13. Note that before we run the tabu search for the scheduling part, we reinitialise the tabu lists for every neighbourhood in the mixed neighbourhood. However, we still keep the information on the probabilities of selecting a neighbourhood in each iteration. The algorithm stops after the main iteration reaches N_{main}.

Algorithm 18 Tabu Search for solving the MPSDP

Input: $\mathfrak{D}^0$
Parameter: N_{Dis}, N_{Sch}, N_{Main}
Initial:
$it := 1$
1: **for all** $i \in D$ **do**
2: Run Algorithm 14 on solution $\mathfrak{D}^{0,i}$ for N_{Sch} sub-iterations ▷ Improve the scheduling part of the initial solution
3: **end for**
4: $\mathfrak{D}^* := \mathfrak{D}^0$ ▷ Update the best neighbouring solution
5: **while** $it < N_{main}$ **do**
6: Find the best non-tabu solution $\mathfrak{D}'$ in Algorithm 17 for at most N_{Dis} sub-iterations ▷ Run tabu search for the districting part
7: **for all** $i \in D$ **do**
8: Run Algorithm 14 on solution $\mathfrak{D}'^{,i}$ for N_{Sch} sub-iterations ▷ Run tabu search for the scheduling part
9: **end for**
10: **if** $z(\mathfrak{D}') < z(\mathfrak{D}^*)$ **then**
11: $\mathfrak{D}^* := \mathfrak{D}'$ ▷ Update the best neighbouring solution
12: **end if**
13: $it := it + 1$ ▷ Increment the number of the main iterations
14: **end while**
Output: $\mathfrak{D}^*$

5.2 Data Generation and Programme Set-Up

All of the results presented in this section are derived from the same machine that performed the experiments in Chapters 3 and 4.

We generate random large data instances that contain a group of 100, 200 and 300 customers. For each group of customers, there are three sets of possible week rhythms with a different planning horizon, which are the same as in Section 2.5. As a result, we have nine combinations of the number of customers and the set of possible week rhythms. We use the same notation to call each combination. For every combination, we generate 5 data instances randomly so there are 45 instances in total in the experiment. Similarly in Chapter 4, for the tabu search algorithm, we collect five repetitions of the experiment per data instance to obtain more reliable statistical results. Therefore, there are 25 results per combination and 225 results in total. Note that a large number of customers makes the problem significantly more difficult to solve.

For each instance, we derive the information on every customer, i.e., the week rhythm, the constant service time, and the location, in the same way as in the previous experiments: see

Section 2.5 for more details. The additional information for this experiment is the locations of salesmen. We assume that one salesman is responsible for around 50 customers. Therefore, there are two, four, and six offices for salesmen for the data sets with 100, 200, and 300 customers, respectively. The two coordinates that represent the locations of the salesmen's offices follow an independent continuous uniform distribution between 2 and 8. This is to prevent any of the offices from being located too close to the boundary of the area. Regarding the maximum deviation from the workload on the district level and the on week level, the values of τ^{dis} and τ^{week} are 0.02 and 0.1, respectively. The weight of the compactness on the district level (λ) is 0.5, which means we equally emphasise the compactness on both levels.

In this experiment, we set the results from the Benders' decomposition of CPLEX as our benchmark, since it is the exact solution approach that performed best overall for the more complicated data instances (see Chapter 3). We set up CPLEX to run on a single thread with no pre-processing. The satisfactory tolerance level between the bounds of the objective value is 0.01%, which is the default value for CPLEX. The maximum computational time for each data instance is 3600 seconds (1 hour).

Regarding the extended tabu search, the stopping criterion for the main algorithm is 20 main iterations (i.e., N_{Main} is 20). For the tabu search of the districting part, the maximum number of sub-iterations, N_{Dis}, is 10. The length of the tabu list T_D is $\frac{N_{Dis}}{2}$. For every customer, the positive parameter γ to limit the number of surrounding customers is 0.2. In the case of the reduced search in the districting neighbourhood, the parameter $\epsilon \in \{0.1, 0.3, 0.5\}$.

For the tabu search of the scheduling part in district $i \in D$, N_{Sch} is set to 100 for the stopping criterion. We use the mixed neighbourhood from Chapter 4 which consists of three neighbourhoods: the week centre neighbourhood with QAP, the switching neighbourhood with the aspiration criterion, and the swapping neighbourhood with the aspiration criterion and the 30% size-reduced neighbourhood. The two tabu lists in the mixed neighbourhood still have the same length as in Chapter 4. Finally, let $|T_{WC}|$ and $|T_{WP}|$ be the length of the tabu list for the week centre neighbourhood and the week pattern neighbourhoods respectively, i.e.,

$$|T_{WC}| = \left\lfloor \frac{|W|}{2} \right\rfloor,$$
$$|T_{WP}| = \left\lfloor \frac{1}{4} \sum_{b \in \mathfrak{B}^i_{NW}} r_b \right\rfloor$$

where $\mathfrak{B}^i_{NW}$ is the set of customer in district i who do not require weekly service. The rest of the parameters in Algorithm 14 are also the same as in the experiment in Chapter 4.

5.3 Computational Results

In this section, we select the Benders' decomposition algorithm in CPLEX as a benchmark for the extended tabu search and show that it struggles to solve data instances with more than 100 customers. Also, we investigate the quality of the initial solutions of the tabu search compared to the benchmark. Then, we show the results of the extended tabu search with different configurations. We present the results of the reduced search in the districting neighbourhood and discuss their performance. We emphasise the advantage of the districting neighbourhood by comparing the results of our algorithm to those of the tabu search without the districting neighbourhood. We then compare our method to the Benders' decomposition algorithm from CPLEX. Finally, we prove the effectiveness and the efficiency of the tabu search algorithm by comparing it to built-in heuristics in CPLEX.

As in Chapter 4, we compare the results in terms of the relative percentage deviation, where positive values imply better performance. We count the number of comparable and better results. Additional information of interest is the average computational time in each combination.

Data	Best %Gap	%Gap	Worst %Gap	Max #Better	Total time (s)
100_1	0.01	1.18	3.39	20	3316.2
100_2	10.41	17.93	22.69	25	3615.8
100_3	48.23	51.44	54.51	25	3629.2
200_1	x	x	x	25	3610.1
200_2	x	x	x	25	3602.8
200_3	x	x	x	25	3600.2
300_1	x	x	x	25	3610.5
300_2	x	x	x	25	3615.3
300_3	x	x	x	25	3630.1
Total				220	

Table 5.1: The information about the best-found solutions by the Benders' decomposition algorithm in CPLEX.

5.3.1 Benchmark for Tabu Search

Table 5.1 presents the results from the Benders' decomposition algorithm in CPLEX, which will be a benchmark for the extended tabu search. For each combination, we present information for the relative percentage gap between the lower bound and the upper bound on the objective value, where Columns *Best %Gap*, *%Gap*, and *Worst %Gap* show, respectively, the best, the average, and the worst relative percentage gap among the 5 data instances. As the method cannot find any feasible solution within the given time for any instance with more than 100 customers, we do not have any information for those cases. Therefore. we print an 'x' in these columns. Column *Max #Better* shows the number of results that find a better solution. This is limited by the number of instances that can find the optimal solution. The last column, *Total time (s)*, presents the average computational time in seconds for each combination.

The table shows that under the limitation of time the method can find feasible solutions only for instances with 100 customers. Also, it can find the optimal solution only in one instance of 100_1. As every instance has 5 results from repetitions, we cannot find better solutions in 5 results out of 25 so *Max #Better* in this case is 20. For the rest of the combinations with 100 customers, the relative percentage gaps are high, especially for 100_3. For every instance with 200 and 300 customers, the exact solution approach cannot find any feasible solution within 1 hour. Therefore, if our method can find any feasible solution under the same limitation of time, it means that our method outperforms the exact solution approach. The number in the last row of *Max #Better* is the maximum possible total number of results that can be better than the benchmark.

5.3.2 Comparison between the Initial Solutions and the Benchmark

Before we investigate the effectiveness of the tabu search algorithm, we check the quality of the initial solutions first. In this section, we check the solution quality in terms of the relative percentage gaps compared to the benchmark from CPLEX. Here, we focus on the compactness on the district level and on the week level to see the solution quality more closely. Then, we check the objective function value for the overall performance.

We can derive an initial solution for every data instance within a second, even for data instances that contain 200 and 300 customers. In those cases, we conclude immediately that our initial solution is better, as the benchmark is not able to derive any solution. Therefore, here we will focus on only the performance of the algorithm for the data instances with 100 customers.

Tables 5.2 and 5.3 show the comparison in terms of the compactness on the district level and on the week level, respectively. For each combination, we focus on the worst, the average, and the best percentage gap among 25 results, as shown in Columns *Worst %Gap**, *%Gap**, and *Best*

Data	District Level				
	Worst %Gap*	**%Gap***	**Best %Gap***	**#Good (75)**	**#Better (75)**
100_1	0.12	6.11	16.60	25	25
100_2	3.47	6.76	16.85	25	25
100_3	3.97	9.82	18.88	25	25
Total				75	75

Table 5.2: The performance of the initial solutions in terms of the compactness on the district level for the data instances with 100 customers.

*%Gap**, respectively. The number of comparable results and that of better results compared to the benchmark are presented in Columns *#Good (75)* and *#Better (75)*, respectively. The number in brackets shows the maximum possible total number of results in each aspect. Note that it is possible to get better compactness on the district level or on the week level in every case so the maximum total number of cases for the better results is the total number of results for the data instances with 100 customers.

In terms of the compactness on the district level, our initial solution is better in every case: see the positive values of *Worst %Gap** and the total of *#Better*. However, the quality on the week level is the opposite for less complicated combinations; for 100_1 and 100_2, the corresponding negative values in *%Gap** shows that the initial solution tends to be worse than the benchmark. Moreover, the number of *#Good* and *#Better* in these two combinations are less than a fifth of the corresponding possible cases. For 100_3 which is the most complicated combination in the consideration here, the compactness on the week level from the initial solutions outperforms the benchmark in every possible case.

To see the overall performance, Table 5.4 compares the initial solutions to the benchmark in terms of the objective value. In this table, the number of results where the initial solutions are better than the benchmark is shown in Sub-column *Initial* of Columns *#Better*. Here, the possible cases for better results depend on the number of data instances where we find the optimal solutions. Therefore, a limit of each combination is presented in Sub-column *Max* of Columns *#Better*, which is the same as in Column *Max #Better* of Table 5.1. Moreover, we investigate if the initial solutions are the same as the optimal solutions found by the benchmark, as shown in Sub-column *Initial* of Columns *#Opt*. Note that the benchmark can find the optimal solution in only one data instance of the combination 100_1. A s we do not have information about the optimal solutions, except for the combination 100_1, we present 'x' in Columns *#Opt* for such cases.

For 100_1 and 100_2, the negative value of *%Gap** and a small number of *#Good* and *#Better* indicate that the initial solutions of most results are worse than the benchmark. This implies that in those results better performance on the district level cannot compensate for the deteriorating quality on the week level. Moreover, the initial solutions are not optimal in the possible cases of 100_1: see the corresponding number of *#Opt*. However, the initial solutions in 100_3 are better in every case. As a result, we gain comparable and better results for the data instances with 100 customers in, respectively, 45% and 48% of the total number of possible cases. The results show that CPLEX is capable of finding good-quality solutions only for the less complicated instances in the experiment such as 100_1 and 100_2. Nevertheless, in such cases, the initial solutions are comparable or better than the benchmark in less than half of the possible cases.

In conclusion, the initial solutions are quick to derive, even for complicated data instances with 200 and 300 customers where CPLEX cannot find any solution within an hour. However, when we compare the solution quality for the less complicated data instances where CPLEX manages to find good solutions, the initial solutions struggle to outperform the benchmark. Therefore, we will improve the quality of the initial solutions by tabu search, as shown in the next sections.

Data	Week Level				
	Worst %Gap*	**%Gap***	**Best %Gap***	**#Good (75)**	**#Better (75)**
100_1	-37.86	-11.05	0.49	4	4
100_2	-13.87	-4.20	8.67	5	5
100_3	2.52	22.59	31.95	25	25
Total				34	34

Table 5.3: The performance of the initial solutions in terms of the compactness on the week level for the data instances with 100 customers.

Data	Worst %Gap*	%Gap*	Best %Gap*	#Good (75)	#Better		#Opt	
					Initial	**Max**	**Initial**	**Max**
100_1	-19.00	-5.51	0.39	4	4	20	0	5
100_2	-6.55	-1.77	7.97	5	5	25	x	x
100_3	6.42	20.52	28.40	25	25	25	x	x
Total				34	34	70	0	5

Table 5.4: The performance of the initial solutions in terms of the objective value for the data instances with 100 customers.

5.3.3 Effectiveness of Different Size Reductions on the Districting Neighbourhood

Now we investigate the effectiveness of the reduced search in the districting neighbourhood. As we do not have a benchmark from CPLEX for instances with 200 and 300 customers, we compare the results of the reduced search to those of the exhaustive search. In other words, the exhaustive search is a benchmark in this case for calculating the relative percentage gaps. We evaluate the solutions in the same way as in Section 5.3.2, i.e., the compactness on the district level, on the week level, and the objective value.

Table 5.5 shows the relative percentage gaps in terms of the compactness on the district level. ϵ-*D* represents the reduced search in the districting neighbourhood with respect to a value of parameter ϵ. As we compare the results to the tabu search with the exhaustive search, which is a heuristic and does not guarantee the optimality condition, it is possible to find a better solution than the benchmark in every result. Therefore, the maximum total number of *#Better* is also 225, as shown in the brackets of the column header.

Overall, *0.1-D* is superior in every aspect. It has the highest total of *#Good* and *#Better*, which are 90% and 88% of the possible total number of cases, respectively. In particular, when the number of customers increases, *0.1-D* manages to improve the quality of the compactness of districts in almost every result: see Column *#Better* of *0.1-D*. Therefore, the reduced search in the districting neighbourhood improves the compactness on the district level in terms of the relative percentage gaps.

Although the reduced search improves the compactness on the district level, it affects differently the compactness of weekly schedules, as we have seen in Section 5.3.2. In Table 5.6, the relative percentage gaps tend to be worse when we reduce the search space in the districting neighbourhood: note the decreasing number of *#Good* and *#Better*. There is a conflict between the compactness on the district level and on the week level, i.e., a highly compact district does not necessarily result in highly compact weekly schedules. This is not surprising as the compactness of weekly schedules does not only rely on the locations of customers but also their week rhythms.

As we equally emphasise the districting part and the scheduling part, the deteriorating weekly schedules in the reduced search affect the quality of the objective values, as shown in Table 5.7. Although the districting part is improved in the reduced search, it is, again, not

Data	Worst %Gap*			%Gap*			Best %Gap*			#Good (225)			#Better (225)		
	0.5-D	**0.3-D**	**0.1-D**	**0.5-D**	**0.3-D**	**0.1-D**	**0.5-D**	**0.3-D**	**0.1-D**	**0.5-D**	**0.3-D**	**0.1-D**	**0.5-D**	**0.3-D**	**0.1-D**
100_1	-6.98	-1.33	-1.32	-0.21	0.39	1.82	4.76	7.23	13.27	21	9	14	2	8	14
100_2	-4.10	-3.59	-0.46	-0.56	-0.39	3.92	1.09	1.65	15.79	12	15	23	5	11	22
100_3	-4.59	-7.64	-1.22	-0.23	0.30	1.25	4.90	5.64	6.44	10	15	17	7	11	16
200_1	-2.72	0.06	0.43	0.18	1.54	2.82	3.94	4.51	7.61	16	25	25	16	25	25
200_2	-2.21	-0.30	-0.16	0.40	1.25	1.95	1.68	2.74	3.33	19	24	24	19	24	24
200_3	-0.82	-1.20	-0.51	0.42	1.31	2.14	1.45	3.23	4.54	19	22	24	18	22	24
300_1	-4.40	-1.47	0.45	0.18	1.31	2.11	2.91	3.89	4.89	18	20	25	18	20	25
300_2	-0.29	-0.51	0.62	1.02	1.15	2.28	2.79	4.00	5.23	22	21	25	21	21	25
300_3	-0.20	0.04	-1.09	1.19	2.56	2.69	5.24	5.31	5.65	23	25	24	23	25	24
Total										160	176	201	129	167	199

Table 5.5: The performance of the reduced search in the districting neighbourhood in terms of the compactness on the district level.

Data	Worst %Gap*			%Gap*			Best %Gap*			#Good (225)			#Better (225)		
	0.5-D	**0.3-D**	**0.1-D**	**0.5-D**	**0.3-D**	**0.1-D**	**0.5-D**	**0.3-D**	**0.1-D**	**0.5-D**	**0.3-D**	**0.1-D**	**0.5-D**	**0.3-D**	**0.1-D**
100_1	-3.93	-4.86	-12.42	0.09	-0.30	-1.70	3.95	1.31	2.62	21	17	13	7	16	13
100_2	-4.95	-5.42	-13.31	-0.08	-0.48	-1.97	1.46	1.46	8.95	19	14	9	10	11	9
100_3	-6.93	-11.65	-10.11	0.28	-1.16	-1.39	7.45	1.70	4.69	13	12	14	12	8	14
200_1	-6.02	-6.27	-20.88	-1.04	-3.46	-7.67	1.73	-2.04	-0.69	7	0	0	7	0	0
200_2	-16.72	-17.81	-22.73	-3.04	-4.31	-6.25	2.71	-0.24	1.08	5	0	5	5	0	5
200_3	-7.75	-17.91	-24.65	-2.16	-7.10	-11.11	1.06	1.91	-0.57	4	3	0	3	3	0
300_1	-9.95	-14.30	-16.35	-1.97	-4.96	-8.94	6.57	0.62	-1.54	5	1	0	5	1	0
300_2	-15.08	-17.90	-27.74	-3.87	-5.64	-9.24	0.83	-1.11	-3.09	4	0	0	4	0	0
300_3	-7.72	-9.96	-11.78	-3.24	-6.99	-8.47	-0.05	-3.57	-4.47	0	0	0	0	0	0
Total										78	47	41	53	39	41

Table 5.6: The performance of the reduced search in the districting neighbourhood in terms of the compactness on the week level.

enough to compensate for the deterioration of the schedules. Therefore, the overall quality of the results is not high, especially when instances become more challenging. *0.5-D* has the highest total of *#Good* and *#Better* which are, however, only 33% and 20% of the possible total number of cases, respectively.

Table 5.8 shows information for the average computational time, where we highlight the best average time in bold for each combination. Note that *ED* is the extended tabu search with the exhaustive search in the districting neighbourhood. The reduced search tends to spend less amount of time on average for the more complicated combinations, such as 100_3, 200_3, 300_1, and 300_3. For 100_3, the reduced search can improve the average time by 32 - 43%. For the rest of those complicated combinations, the reduced search spend between 2% and 21% less computational time on average. However, it is not clear if the smaller search space in the districting neighbourhood can save more computational time. The tabu search for the districting part is usually finished within a second, so the main computational time of the extended tabu search comes from running the tabu search for the scheduling part in every district sequentially. Therefore, a more effective way to reduce the computational time is to solve the tabu search for the scheduling part of every district in parallel.

Although the reduced search spends less computational time, it is not enough to compensate for the low quality of the objective values. Therefore, we conclude that *ED* has a better performance than the method with the reduced search.

5.3.4 Advantages of the Districting Neighbourhood

Next, we show the benefit of the districting neighbourhood in the extended tabu search. We collect the results of the extended tabu search without the districting neighbourhood. In such an algorithm, we solve the classical sales districting problem to partition customers into districts and then run only the tabu search for the scheduling part. The maximum total number of iterations for the scheduling part is 2000 in every district, which is the same total number of sub-iterations for the scheduling part in every district for the extended tabu search algorithm. We call this algorithm *NoDis*. This algorithm was suggested in Bender et al. (2016) to solve the MPSDP, i.e., solving the districting part and the scheduling part sequentially. We set up *NoDis* as a benchmark for *ED*, which is our best algorithm now, where a positive value for the relative percentage gap indicates that *ED* has superior performance. Again, the maximum total number of *#Better* here is 225 as we compare the results with a heuristic.

The comparison in terms of the compactness on the district level and on the week level is shown in Table 5.9. The districting part of *ED* is worse than the benchmark, i.e., the best solutions of *ED* tend to have less compact districts: note the negative values of the relative percentage gaps in Columns *District Level*. In particular, the total of *#Better* indicates that every result of *ED* does not have a better compactness on the district level. However, *ED* improves the scheduling part significantly: see the positive values of the relative percentage gaps in Columns *Week Level*. Moreover, the total of *#Good* and *#Better* show that more than 98% of the possible cases are improved in the aspect. As a result, *ED* is superior in terms of the objective value, as shown in Table 5.10. At least 98% of the total results achieve better objective values.

In terms of the average computational time which is shown in Table 5.10, both algorithms are quite competitive in the less complicated combinations, e.g., in 100_1, 100_2, and 300_1. However, *ED* spends considerably more time on average in some complicated combinations. For 100_3 and 300_3, *ED* spends 87% and 40% more time, respectively. Nevertheless, the increasing time is still acceptable for deriving high-quality solutions. Therefore, we conclude that *ED* is superior, which proves the advantage of the districting neighbourhood.

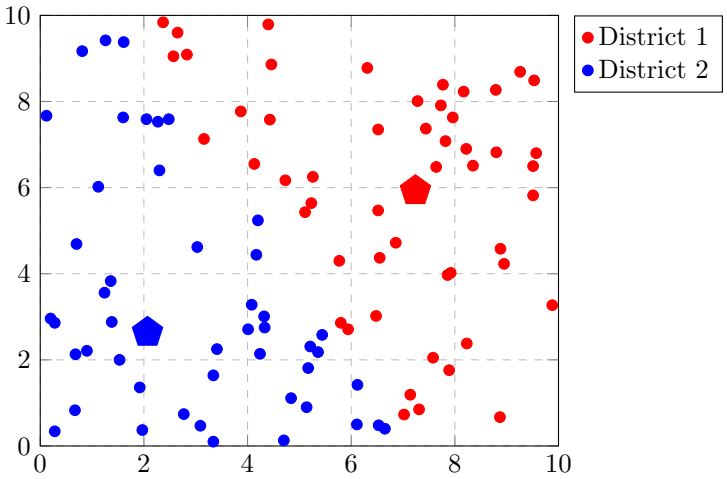
The above results show that we might not get high-quality weekly schedules by solving the districting part and the scheduling part sequentially, as the most compact districts from the classical districting problem do not give guarantees on the quality of compactness for the weekly schedules. Figures 5.1 and 5.2 present districts and one-week schedules from the best solution by both algorithms in an instance of 100_1, respectively. Both algorithms derived districts with different shapes. *NoDis* has slightly better districts as they do not overlap: see Figure 5.1a. However, Figure 5.2b shows that the schedules of *ED* are more compact. For figures of every weekly schedule from this solution, we refer interested readers to Appendix A.

Data	Worst %Gap*			%Gap*			Best %Gap*			#Good (225)			#Better (225)		
	0.5-D	**0.3-D**	**0.1-D**	**0.5-D**	**0.3-D**	**0.1-D**	**0.5-D**	**0.3-D**	**0.1-D**	**0.5-D**	**0.3-D**	**0.1-D**	**0.5-D**	**0.3-D**	**0.1-D**
100_1	-0.92	-0.68	-3.51	0.00	0.00	-0.46	0.42	0.67	1.68	21	10	13	6	9	13
100_2	-3.80	-4.08	-6.18	-0.21	-0.48	-0.46	0.33	0.51	7.41	15	14	9	7	10	9
100_3	-4.00	-7.45	-6.74	0.18	-0.80	-0.73	5.18	0.89	3.94	13	12	14	9	7	14
200_1	-3.55	-3.75	-11.45	-0.61	-1.75	-4.17	0.96	-0.55	-0.16	5	0	0	5	0	0
200_2	-11.61	-12.11	-15.78	-2.10	-2.92	-4.25	2.09	-0.03	0.77	4	0	5	3	0	5
200_3	-4.62	-10.73	-14.80	-1.30	-4.30	-6.72	0.59	1.10	-0.40	5	3	0	5	3	0
300_1	-4.93	-7.20	-9.17	-1.15	-2.75	-5.18	3.21	-0.02	-0.77	5	0	0	5	0	0
300_2	-10.05	-12.05	-18.59	-2.56	-3.86	-6.24	1.02	-0.75	-2.19	4	0	0	4	0	0
300_3	-4.26	-6.01	-7.29	-1.84	-3.98	-4.96	0.09	-2.03	-3.13	1	0	0	1	0	0
Total										73	39	41	45	29	41

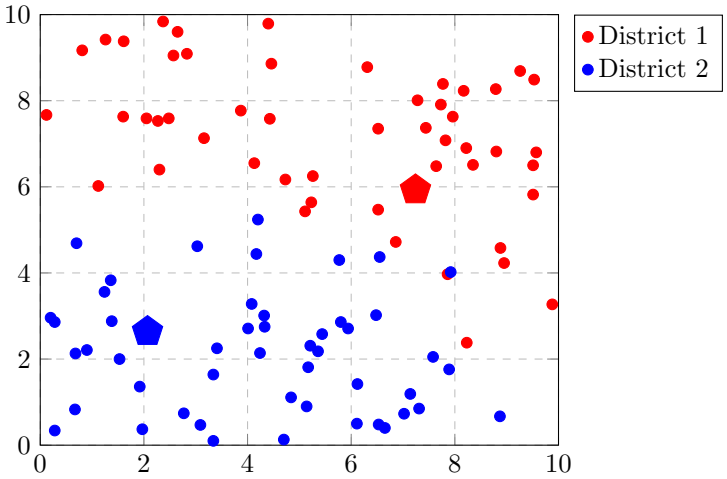
Table 5.7: The performance of the reduced search in the districting neighbourhood in terms of the objective value.

Data	Total time (s)			
	ED	**0.5-D**	**0.3-D**	**0.1-D**
100_1	**60.16**	65.84	74.92	65.04
100_2	83.84	**81.08**	101.76	83.68
100_3	299.04	**168.68**	200.4	175.48
200_1	**158.32**	160.48	199.04	190.24
200_2	224.8	222.8	263.2	**221.2**
200_3	492.2	461.8	**390.4**	437.64
300_1	371.04	364.48	305.8	**300.84**
300_2	**342.48**	397.8	358.04	349.68
300_3	994.2	873.76	844.44	**784.52**

Table 5.8: The average computational time of the different size reductions in the districting neighbourhood.



(a) NoDis



(b) ED

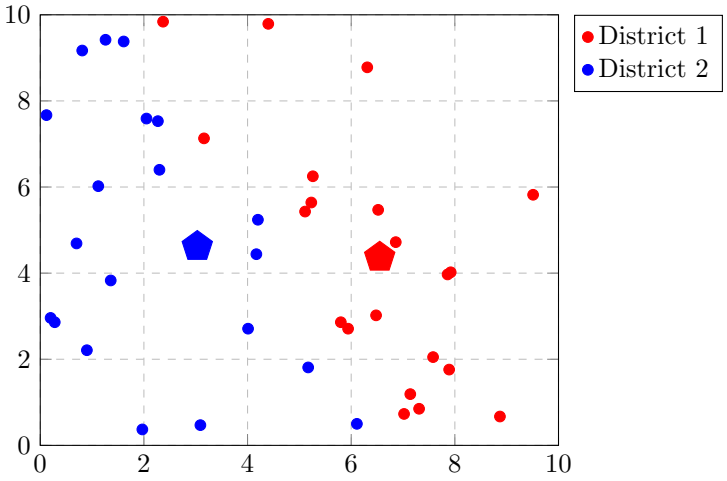
Figure 5.1: The districts from the best solution in an instance of 100_1.

Data	District Level					Week Level				
	Worst %Gap*	**%Gap***	**Best %Gap***	**#Good (225)**	**#Better (225)**	**Worst %Gap***	**%Gap***	**Best %Gap***	**#Good (225)**	**#Better (225)**
100_1	-20.11	-4.28	-0.25	0	0	0.28	6.61	24.39	25	25
100_2	-21.89	-6.19	0.00	1	0	0.00	7.93	17.59	25	24
100_3	-8.23	-2.26	0.00	2	0	-0.09	4.17	11.72	24	22
200_1	-11.91	-4.21	-1.30	0	0	3.84	9.58	24.28	25	25
200_2	-7.01	-2.92	-0.44	0	0	0.24	7.65	19.35	25	25
200_3	-5.64	-3.43	-0.29	0	0	1.40	12.80	27.50	25	25
300_1	-5.49	-2.58	-0.69	0	0	1.92	8.73	14.51	25	25
300_2	-7.19	-3.20	-1.01	0	0	3.17	9.90	25.41	25	25
300_3	-7.64	-4.67	-1.58	0	0	7.32	10.96	15.31	25	25
Total				3	0				224	221

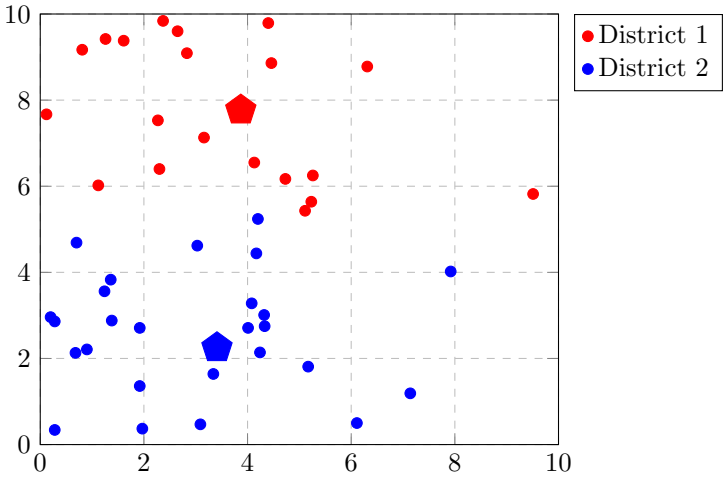
Table 5.9: The performance of the extended tabu search compare to the one without the districting neighbourhood in terms of the compactness on the district level and on the week level.

Data	Worst %Gap*	%Gap*	Best %Gap*	#Good (225)	#Better (225)	Total time (s)	
						NoDis	ED
100_1	0.14	3.85	13.24	25	25	63.56	60.16
100_2	0.00	5.35	10.42	25	24	84.16	83.84
100_3	-0.15	2.78	8.21	24	22	159.72	299.04
200_1	1.90	5.65	15.21	25	25	197.04	158.32
200_2	0.03	5.49	14.14	25	25	217.24	224.8
200_3	0.85	8.33	18.57	25	25	420.68	492.2
300_1	1.01	5.24	8.67	25	25	369.2	371.04
300_2	2.22	7.03	18.50	25	25	353.28	342.48
300_3	4.66	6.57	9.42	25	25	703.96	994.2
Total				224	221		

Table 5.10: The performance of the extended tabu search compared to the one without the districting neighbourhood in terms of the objective value.



(a) NoDis



(b) ED

Figure 5.2: The weekly schedules for week 2 from the best solution in an instance of 100_1.

5.3.5 Improvement on the Initial Solutions by the Tabu Search

We have selected the best configuration of the tabu search algorithm to solve the MPSDP: the exhaustive search in the districting neighbourhood (*ED*). Here, we investigate the improvement

over the initial solutions by the tabu search algorithm. Note that in terms of the objective value, the maximum total number of possible cases for better results is the total number of results since every initial solution has not been proved to be optimal yet.

Table 5.11 shows the comparison in terms of the compactness on the district level and on the week level. It shows the conflict between the districting part and the scheduling part again. The best solutions from the tabu search algorithm have worse compactness on the district level but achieve significantly better quality for the scheduling part. The tabu search algorithm improves the compactness on the week level in every case: see the total of *#Good* and that of *#Better*.

We have shown in Section 5.3.2 and 5.3.4 that the quality of the compactness in the scheduling part mainly influences the objective value. Table 5.12 supports the observation that the objective value of the best solutions by the tabu search algorithm outperforms those of the initial solutions in every case. Note that we are not interested in a number of comparable solutions as the algorithm accepts only an improved solution from the initial solutions.

In conclusion, the results illustrate that the tabu search algorithm is an effective solution improvement technique.

5.3.6 Comparison between the Tabu Search and the Benchmark

Now, we compare solutions from *ED* to those of the Benders' decomposition in CPLEX for instances with 100 customers. For the rest of the combinations, we immediately conclude that the tabu search algorithm is more effective as we can derive a solution in every case.

Tables 5.13 and 5.14 show the comparison in terms of the compactness on the district level and on week level, respectively. Note that it is still possible to have better compactness in the districting part or in the scheduling part, so the maximum total number of cases for better results is 75 (the total number of results of the data instances with 100 customers). In terms of the districting part, for the combinations 100_1 and 100_2, *ED* finds comparable and better results at least in half of the possible cases. For 100_3, the algorithm works well as a solution from the tabu search is better in every possible case. In terms of the scheduling part, *ED* does not perform considerably well in 100_1: see a small number of *#Good* and *#Better*. However, *ED* performs well in the rest of 100 instances. Overall, the total of *#Good* and *#Better* reach at least 76% of the possible total number of cases.

Table 5.15 shows information related to the relative percentage gaps of the objective value. Again, we shows the information on the number of results that are comparable or better than the benchmark. In Columns *#Better*, the number of better results by the tabu search algorithm is presented in Sub-column *ED*. Since the benchmark can find the optimal solution in one data instance (shown in Section 5.3.1), Sub-column *Max* shows a limit of the number of better results in each case, which is the same as Column *Max #Better* in Table 5.1. Moreover, we check if any best solution by our algorithm is optimal in Columns *#Opt*, where the information of the maximum possible cases in Sub-column *Max* is the same as of Sub-column *Max* in Columns *#Opt* of Table 5.4.

In terms of the objective value, *ED* struggles to find good and better results in 100_1: note a small number of *#Good* and *#Better*. Furthermore, the number of *#Opt* confirms that we still cannot find the optimal solution in the possible cases. For the rest of the combinations, *ED* outperforms the benchmark. The positive values in *Worst %Gap** in those cases indicate that *ED* improves the objective values in every result.

Importantly, the extended tabu search finds solutions for every instance in a reasonable amount of time, which is less than 17 minutes: see Table 5.16. Apart from 300_3, the heuristic spends at least 86% less time than CPLEX. For 300_3 which is the most challenging instance, the extended tabu search can still reduce the time by 70% which is a considerable amount.

The above results in terms of the solution quality and the computational times prove the robustness of the heuristic to solve the MPSDP, especially where the benchmark cannot find any solution within an hour.

Data	District Level					Week Level				
	Worst %Gap*	**%Gap***	**Best %Gap***	**#Good (225)**	**#Better (225)**	**Worst %Gap***	**%Gap***	**Best %Gap***	**#Good (225)**	**#Better (225)**
100_1	-20.11	-4.28	-0.25	0	0	0.31	7.39	27.53	25	25
100_2	-21.89	-6.19	0.00	1	0	0.12	8.22	17.62	25	25
100_3	-8.23	-2.26	0.00	2	0	1.33	8.23	14.32	25	25
200_1	-11.91	-4.21	-1.30	0	0	4.36	10.71	26.57	25	25
200_2	-7.01	-2.92	-0.44	0	0	0.49	8.12	21.14	25	25
200_3	-5.64	-3.43	-0.29	0	0	3.66	16.77	31.98	25	25
300_1	-5.49	-2.58	-0.69	0	0	3.29	9.48	15.95	25	25
300_2	-7.19	-3.20	-1.01	0	0	3.70	10.29	25.98	25	25
300_3	-7.64	-4.67	-1.58	0	0	8.33	14.44	21.78	25	25
Total				3	0				225	225

Table 5.11: The improvement on the initial solutions by the tabu search algorithm in terms of the compactness on the district level and on the week level.

Data	Worst %Gap*	%Gap*	Best %Gap*	#Better (225)
100_1	0.17	4.45	15.97	25
100_2	0.10	5.58	10.53	25
100_3	1.00	6.02	10.16	25
200_1	2.35	6.45	17.19	25
200_2	0.21	5.86	15.68	25
200_3	2.61	11.39	22.34	25
300_1	1.94	5.79	9.76	25
300_2	2.64	7.35	18.99	25
300_3	5.42	9.23	14.21	25
Total				225

Table 5.12: The overall improvement on the initial solutions by the tabu search algorithm.

Data	District Level				
	Worst %Gap*	%Gap*	Best %Gap*	#Good (75)	#Better (75)
100_1	-0.17	2.46	8.78	18	17
100_2	-1.36	1.37	4.25	19	16
100_3	1.62	7.86	16.07	25	25
Total				62	58

Table 5.13: The performance of the extended tabu search compared to the Benders' decomposition method in CPLEX in terms of the compactness on the district level for the data instances with 100 customers.

Data	Week Level				
	Worst %Gap*	%Gap*	Best %Gap*	#Good (75)	#Better (75)
100_1	-6.92	-1.77	0.81	8	7
100_2	1.25	4.74	8.93	25	25
100_3	15.55	29.11	41.50	25	25
Total				58	57

Table 5.14: The performance of the extended tabu search compared to the Benders' decomposition method in CPLEX in terms of the compactness on the week level for the data instances with 100 customers.

Data	Worst %Gap*	%Gap*	Best %Gap*	#Good (75)	#Better		#Opt	
					ED	Max	ED	Max
100_1	-2.33	-0.52	0.57	8	5	20	0	5
100_2	1.49	4.06	8.10	25	25	25	x	x
100_3	15.68	25.37	35.37	25	25	25	x	x
Total				58	55	70	0	5

Table 5.15: The performance of the extended tabu search compared to the Benders' decomposition method in CPLEX in terms of the objective value for the data instances with 100 customers.

Data	Total time (s)	
	ED	**AutoBD**
100_1	60.16	3316.2
100_2	83.84	3615.8
100_3	299.04	3629.2
200_1	158.32	3610.1
200_2	224.8	3602.8
200_3	492.2	3600.2
300_1	371.04	3610.5
300_2	342.48	3615.3
300_3	994.2	3630.1

Table 5.16: The average computational time of the tabu search compared to those of the Benders' decomposition method in CPLEX.

5.3.7 Comparison between the CPLEX's Heuristics and the Tabu Search

Finally, we compare the efficiency of the tabu search algorithm to the built-in heuristics in CPLEX. Similarly to Section 4.6.10 of Chapter 4, we compare solutions generated by the feasibility pump to the initial solutions. Then, we focus on the performance of RINS (Relaxation Induced Neighbourhood Search) and the solution polishing compared to the tabu search algorithm. Note that our solutions are the benchmark in this section.

Comparison between Solutions from the Feasibility Pump and the Initial Solutions

As discussed before in Section 4.6.10, we choose the feasibility pump to get a feasible solution without taking the objective value into account. The maximum time for running the CPLEX heuristic is 900 seconds (15 minutes). As every initial solution is not optimal, there is still a chance that the feasibility pump can derive a better solution. Therefore, the maximum total number of possible cases for better results is the total number of results.

Unfortunately, the feasibility pump can derive solutions within the limitation of time for only 100_1. Therefore, we will show only the performance of the data instances with 100 customers, where 'x' stands for unavailable information. Again, for every data instance in every combination except 100_1, we conclude immediately that the method to derive the initial solutions outperforms the feasibility pump as it can generate a solution in every case within only a second.

Tables 5.17 and 5.18 show the comparison in terms of the compactness on the district level and on the week level, respectively. In 100_1, the solution quality on the district level is worse in every result: see the negative value of *Best %Gap**. On the week level, the feasibility pump has only 5 better results out of 25 in such combination. As a result, most cases have a worse objective value, as shown in Table 5.19. Moreover, the feasibility pump spends full time on average in every data instance: see Column *FP Time (s)* of Table 5.20.

The feasibility pump struggles to derive any feasible solution for the large data instances, while our method to derive an initial solution requires less than one second and, at the same time, provides better overall solution quality. Therefore, we conclude that our method to derive an initial solution is significantly more effective and efficient.

Comparison between RINS, the Solution Polishing, and the Tabu Search

Similarly to Section 4.6.10, RINS and the solution polishing get the same initial solution as the tabu search at the beginning. The maximum time here is one hour. To get a fair comparison, we limit the maximum number of times to apply the CPLEX heuristics. In the extended tabu search, there are 20 main iterations where each main iteration contains at most 10 sub-iterations for the districting part and 100 sub-iterations for the scheduling part in each district. We set

Data	District Level				
	Worst %Gap*	**%Gap***	**Best %Gap***	**#Good (75)**	**#Better (75)**
100_1	-16.86	-9.34	-0.90	0	0
100_2	x	x	x	x	x
100_3	x	x	x	x	x
Total				0	0

Table 5.17: The performance of the feasibility pump compared to the tabu search method in terms of the compactness on the district level for the data instances with 100 customers.

Data	Week Level				
	Worst %Gap*	**%Gap***	**Best %Gap***	**#Good (75)**	**#Better (75)**
100_1	-256.46	-85.18	14.78	5	5
100_2	x	x	x	x	x
100_3	x	x	x	x	x
Total				5	5

Table 5.18: The performance of the feasibility pump compared to the tabu search method in terms of the compactness on the week level for the data instances with 100 customers.

Data	**Worst %Gap***	**%Gap***	**Best %Gap***	**#Good (75)**	**#Better (75)**
100_1	-179.92	-63.71	9.54	5	5
100_2	x	x	x	x	x
100_3	x	x	x	x	x
Total				5	5

Table 5.19: The performance of the feasibility pump compared to the tabu search method in terms of the objective value for the data instances with 100 customers.

Data	**FP Time (s)**
100_1	915.20
100_2	908.20
100_3	913.40
200_1	907.45
200_2	901.35
200_3	903.26
300_1	912.32
300_2	905.45
300_3	908.21

Table 5.20: The average computational time of the feasibility pump.

Data	Worst %Gap*		%Gap*		Best %Gap*		#Good (75)		#Better (75)	
	RINS	**solPol**	**RINS**	**solPol**	**RINS**	**solPol**	**RINS**	**solPol**	**RINS**	**solPol**
100_1	34.86	30.66	44.34	42.48	51.76	51.60	25	25	25	25
100_2	x	x	x	x	x	x	x	x	x	x
100_3	x	x	x	x	x	x	x	x	x	x
Total							25	25	25	25

Table 5.21: The performance of RINS and the solution polishing compared to the tabu search algorithm in terms of the compactness on the district level for the data instances with 100 customers.

Data	Worst %Gap*		%Gap*		Best %Gap*		#Good (75)		#Better (75)	
	RINS	**solPol**	**RINS**	**solPol**	**RINS**	**solPol**	**RINS**	**solPol**	**RINS**	**solPol**
100_1	-16.08	-5.49	-3.99	0.69	0.18	6.61	3	19	2	17
100_2	x	x	x	x	x	x	x	x	x	x
100_3	x	x	x	x	x	x	x	x	x	x
Total							3	19	2	17

Table 5.22: The performance of RINS and the solution polishing compared to the tabu search algorithm in terms of the compactness on the week level for the data instances with 100 customers.

up the maximum number of nodes for the solution polishing to the maximum total number of sub-iterations in the tabu search. For RINS, we activate the heuristic every 20 nodes, so the maximum number of nodes, in this case, is the maximum total number of sub-iterations multiplied by 20.

RINS and the solution polishing can provide solutions only in the combination 100_1. This implies that these heuristics cannot finish the process to improve a solution within an hour even if they have the same initial solution as the tabu search. In this section, we will show the performance of the algorithms for only the data instances with 100 customers.

Table 5.21, 5.22, and 5.23 present, respectively, the comparison in terms of the compactness on the district level and on the week level, and the objective value. *RINS* and *SolPol* stand for the related performance by RINS and the solution polishing, respectively. Finally, Table 5.24 presents the average time, where *ED* stands for information on the tabu search.

If we focus only the performance of the CPLEX's heuristics in 100_1, it is clear that *solPol* is better than *RINS*: see the corresponding numbers of *#Good* and *#Better* in every Table. Therefore, in this case, the solution polishing is better than RINS. This reinforces the fact that the solution polishing is suitable to find a good solution for a difficult problem (*IBM ILOG CPLEX Optimization Studio CPLEX User's Manual* 2017).

Now, we focus on the performance of the solution polishing compared to the tabu search in 100_1. *solPol* achieves better compactness on the district level and on the week level in every case and 68% of the possible cases, respectively. Overall, it derives a better solution than our algorithm in 60% of the possible cases. However, it requires a full hour, while our algorithm can finish its process within a minute: see Table 5.24.

The above results show that CPLEX's heuristics struggle to provide a high-quality solution for large data instances within an hour. Although the solution polishing can derive a better solution than the tabu search, its capability is limited, i.e., it can solve only the least complicated combination in the experiment and requires much more time. As the tabu search can derive a solution in every case within a reasonable amount of time, i.e., less than 17 minutes, it is incredibly more effective and efficient.

5.3.8 Conclusion

In conclusion, the extended tabu search was successful in solving the MPSDP for instances with 100−300 customers in a reasonable amount of time. Moreover, it is more effective and efficient than the built-in heuristics in CPLEX. The results also support the conclusion that

Data	Worst %Gap*		%Gap*		Best %Gap*		#Good (75)		#Better (75)	
	RINS	**solPol**	**RINS**	**solPol**	**RINS**	**solPol**	**RINS**	**solPol**	**RINS**	**solPol**
100_1	-7.41	-3.60	-2.22	0.10	0.00	2.37	1	19	0	15
100_2	x	x	x	x	x	x	x	x	x	x
100_3	x	x	x	x	x	x	x	x	x	x
Total							1	19	0	15

Table 5.23: The performance of RINS and the solution polishing compared to the tabu search algorithm in terms of the objective value for the data instances with 100 customers.

Data	Total time (s)		
	ED	**RINS**	**SolPolish**
100_1	60.16	3600.12	3600.28
100_2	83.84	3600.81	3600.27
100_3	299.04	3600.89	3601.17
200_1	158.32	3600.63	3600.66
200_2	224.8	3600.41	3600.83
200_3	492.2	3600.14	3600.99
300_1	371.04	3600.73	3601.45
300_2	342.48	3601.80	3602.55
300_3	994.2	3600.31	3602.31

Table 5.24: The average computational time of the tabu search, RINS and the solution polishing.

it is beneficial to include the districting neighbourhood in the method, as there is a conflict between the compactness of the districting part and that of the scheduling part. In particular, the most compact districts from the classical sales districting problem do not guarantee the quality of the compactness of weekly schedules. The main reason is that the compactness of weekly schedules also depends on customers' week rhythm, not only their locations. Therefore, it would be useful to consider both parts at the early stage of planning to ensure high-quality weekly schedules. Since we have just developed this heuristic, there is potential for further improvement; we will discuss this in the next chapter.

Chapter 6

Conclusion and Further Study

6.1 Conclusion

In this work, we studied a recent extension of the classical sales districting problem called the multi-period sales districting problem. For this problem, customers require recurring service with specific frequency in a fixed planning horizon. In addition to partitioning the customers into compact and balanced districts for the salesmen, it is also required to determine weekly schedules for the salesmen to visit customers in a way that satisfies the customers' visiting frequency. The schedules should result in compact and balanced week clusters as these attributes, especially compactness, provide benefits for salesmen: they provide more flexibility for them to make small changes to their schedule during the week without deteriorating their overall performance. In particular, a salesman can fail to serve some customers on a specific day of the original plan due to some unexpected situations, for example, traffic jam or road maintenance. In that case, the compactness of the week cluster ensures that when they postpone the appointments of customers to another day in the week, the overall travel time does not significantly increase compared to the original plan. Although this problem is encountered regularly in sales promotion at customers' sites or engineering field maintenance, it has only been addressed recently and more studies are still required.

The MPSDP can be separated into two parts: the districting part which aims to assign customers to salesmen, as in the classical sales districting problem, and the scheduling part that creates compact and balanced weekly schedules to serve customers in a district. Although an integrated solution approach is desirable, it is challenging to design for solving a large number of customers in realistic instances. Moreover, both parts are required to be updated on different time scales: the districting part is usually updated after several years, while schedules are updated after several months. Since schedules are required to be updated frequently and most previous studies did not consider geographical compactness when creating schedules, the scheduling part of the MPSDP has become a more active area of research in recent years. Moreover, this new line of research suggests solving the MPSDP by solving the districting part (which is the classical sales districting problem) and the scheduling part sequentially.

We first focused on only solving the scheduling part of the problem. Here, we proposed a Benders' decomposition as an exact solution approach and a tabu search algorithm as a metaheuristic. Then, we tested their effectiveness on generated small data instances that contain $30-50$ customers.

For the exact solution approach, we developed the branch-and-Benders' cut algorithm including various sophisticated techniques. At the root node, we added Benders' cuts generated from fractional solutions to strengthen the LP relaxation of the problem. Also, we derived a high-quality integer feasible solution from the location-allocation heuristic to further generate more Benders' cuts and improve the upper bound of the tree. We implemented optimal Pareto cuts to get the strongest Benders' cuts. Moreover, we exploited the decomposable structure of the problem to generate multiple Benders' cuts each time and considerably improve the lower bound of the tree. However, when we compared it to the Benders' decomposition algorithm in CPLEX, it reached the worse relative percentage gaps between the upper bound and the lower

bound. We then investigated more thoroughly the superior performance of CPLEX and found that our developed algorithm struggled to reach a higher lower bound at the end of the tree search. Nevertheless, it could find a better integer solution when data instances became more challenging, so our algorithm is highly competitive to the Benders' decomposition in CPLEX.

Regarding the tabu search algorithm, we introduced three different neighbourhoods based on the information of week centres or customers' week patterns. In each neighbourhood, we proposed techniques to speed up the search. These included allowing infeasible solutions in the search, limiting the search space to promising areas, and using a surrogate objective function when the real objective value is too time-consuming to compute. A high-quality initial solution from the location-allocation heuristic was used to start the search. Also, whenever the search seemed to be trapped around a local optimum, the diversification was run to help in such cases. We integrated these three neighbourhoods in one algorithm to exploit their specific search trajectory. In each iteration, we selected the appropriate neighbourhood by the adaptive probabilities. The algorithm was successful in finding high-quality solutions, while spending much less time compared to the Benders' decomposition algorithm and the built-in heuristics in CPLEX, especially in more challenging data instances. The tabu search algorithm was also employed to enhance the upper bound of our developed Benders' decomposition algorithm. However, the combining of these two methods still requires a more elaborate design to improve efficiency.

Due to the huge success of the tabu search for the scheduling part in small data instances, we extended the method to solve the MPSDP, i.e., to solve the districting and the scheduling part simultaneously. For the tabu search of the districting part, we developed a districting neighbourhood to exchange customers between districts. We tested the extended tabu search on generated large data instances that contain $100-300$ customers and compared the performance to the Benders' decomposition and the built-in heuristics in CPLEX. The results showed that the exact solution approach and the CPLEX heuristics could not even find any feasible integer solution within an hour for any data instances with more than 100 customers. At the same time, the extended tabu search was capable of deriving a high-quality solution within a reasonable amount of time, i.e., less than 17 minutes for every data instance. Moreover, the results showed that the suggestion from previous studies to solve the MPSDP by sequentially solving the districting part and the scheduling part might not be effective since highly compact districts do not ensure highly compact schedules. The main reason is that the compactness of weekly schedules also depends on customers' week rhythm, not only their locations. Therefore, we would suggest considering both parts at the early stage of the planning to ensure high-quality weekly schedules.

Studies of the multi-period sales districting problem are still scarce in the literature, so there is scope for further research. We discuss this in the next section.

6.2 Further Study

In this work, we assumed that every customer requests only one visit per visiting week and daily schedules might not be essential. Also, as it is usual to reschedule visits of customers to another day due to their short-term requests (Bender et al. 2016), we focused only on weekly schedules in the scheduling part. However, it is worth studying under the same planning requirements as in Bender et al. (2016), i.e., customers can request more than one visit per visiting week and might have a preference on a visiting day(s). In that case, daily schedules should be included in the plan. Moreover, it is interesting to add more realistic planning requirements into the problem, for example, uncertainty on customers' requests.

Regarding the proposed methods for solving the scheduling part, it would be useful to test them on real-world data instances to see their actual performance. Moreover, there is potential to further develop the Benders' decomposition and the tabu search algorithms as follows.

For Benders' decomposition, we should adopt the techniques from Bender et al. (2018) to reduce the symmetry of solutions during the branching process. This significantly reduces the size of the branch-and-bound tree and, as a result, accelerates the efficiency of the algorithm. Moreover, we can consider a new decomposition of the problem: keeping only the week pattern variables in the master problem. The main reason is that when we know the week pattern of

every customer, we can manually derive the values of the centre variables and the allocation variables, as in the location problem in the location-allocation heuristic. However, the subproblem which has binary centre variables, in this case, is not a linear programme anymore and we cannot generate the optimality cuts from the dual variables as usual. Therefore, more sophisticated methods to generate cuts from a mixed-integer programming subproblem are required. A potential way is combinatorial cuts (Codato & Fischetti 2006, Gendron et al. 2016). Another interesting method is the nested Benders' decomposition algorithm where the Benders' decomposition algorithm is applied more than once (Naoum-Sawaya & Elhedhli 2010). The last promising technique for us so far is an integer L-shaped method. Note that this technique is also the Benders' decomposition algorithm but the only difference is that it is in the context of stochastic programming. An optimality cut derived from this method, however, has a weakness: it relies on only a master solution that is currently cut off, resulting in a potentially weak approximation of the projected cost. Therefore, Angulo et al. (2016) propose a modified optimality cut that includes information on previously found master solutions to improve the quality of the projected cost.

Regarding the tabu search, it is possible to add more advanced features, for example, adaptive memory (Rochat & Taillard 1995). Although we have proposed some solution approaches for the scheduling part, more sophisticated techniques, especially to tackle large data instances, are still required.

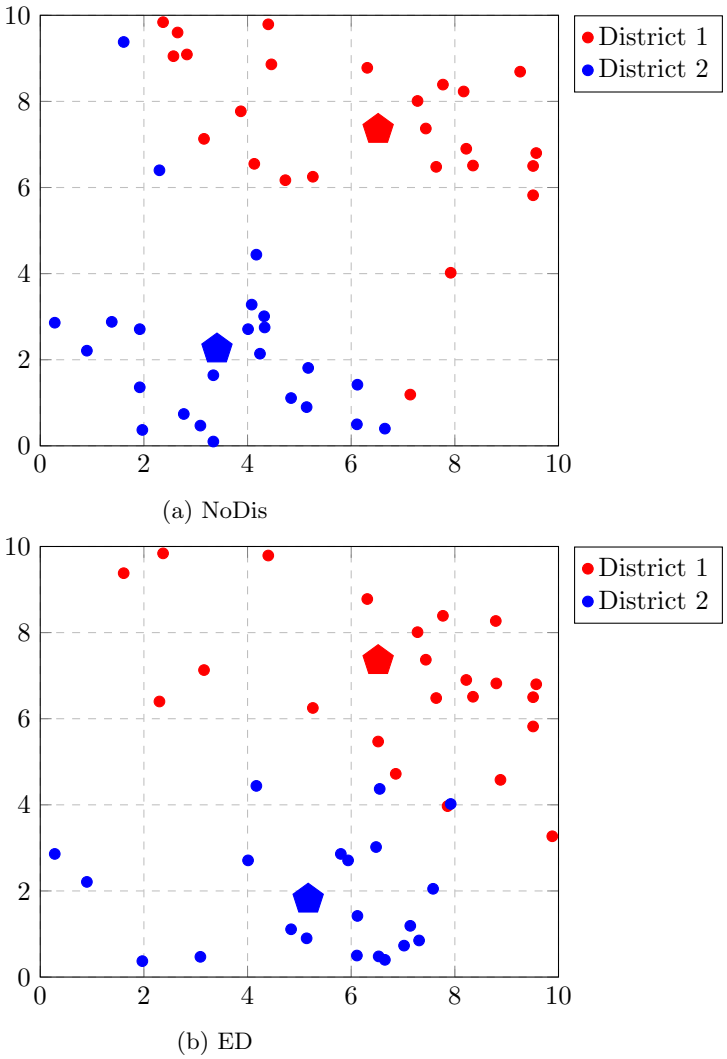
Although the extended tabu search showed an impressive performance in the large data instances, it still requires further development. In particular, it lacks the diversification scheme, which is an essential part of the tabu search. An effective way to implement this can be adopted from Bozkaya et al. (2003) which has proved successful in several papers. Moreover, we can speed up the algorithm easily by running the tabu search for the scheduling part for every district in parallel.

Finally, it is desirable to design more solution approaches for solving the MPSDP, as it is beneficial to solve the districting part and the scheduling part simultaneously. An interesting method for us now is Greedy Randomised Adaptive Search Procedure, since it is one of the most popular meta-heuristic in the districting problem (Kalcsics & Ríos-Mercado 2019).

Appendix A

Figures from the Best Solution

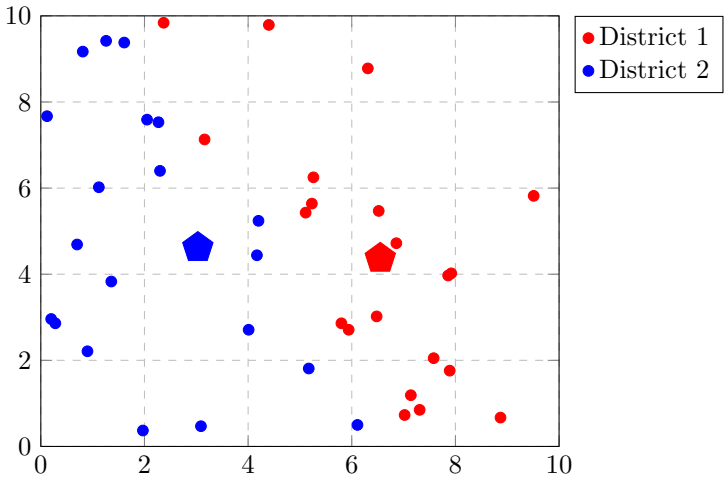
A.1 The Best Solution in Figures 5.1 and 5.2



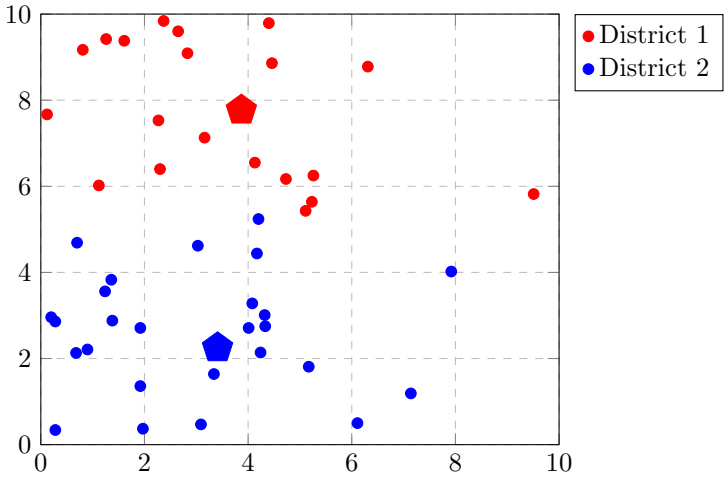
(a) NoDis

(b) ED

Figure A.1: The weekly schedules for week 1 from the best solution in an instance of 100_1.

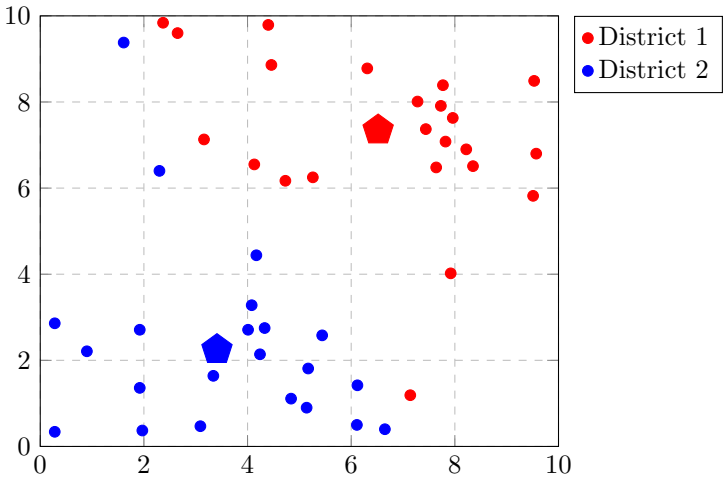


(a) NoDis

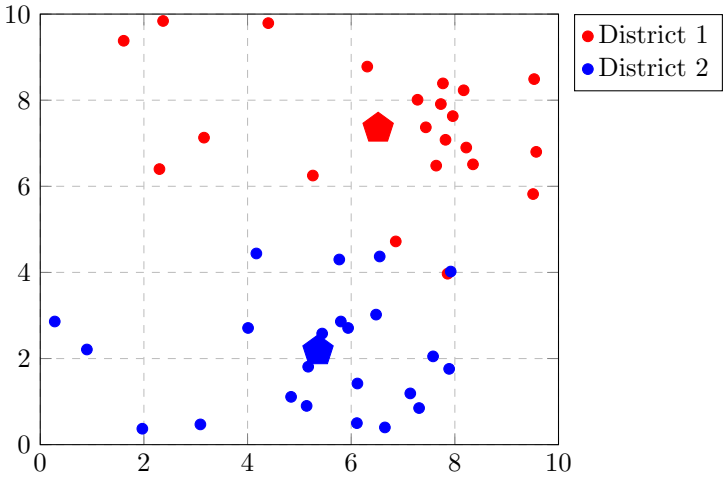


(b) ED

Figure A.2: The weekly schedules for week 2 from the best solution in an instance of 100_1.

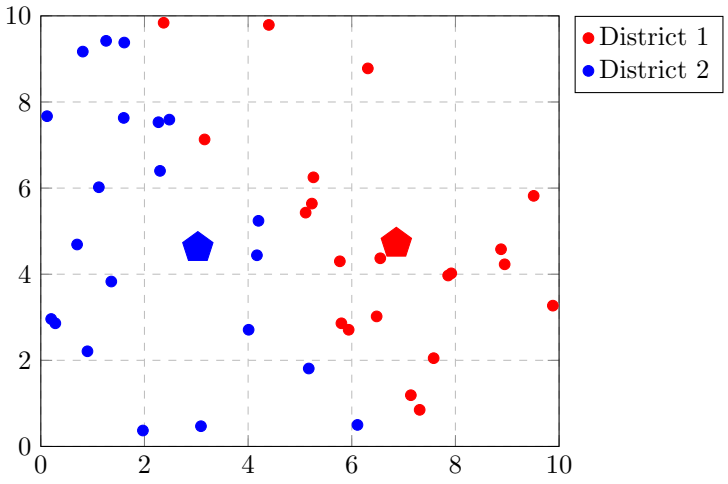


(a) NoDis

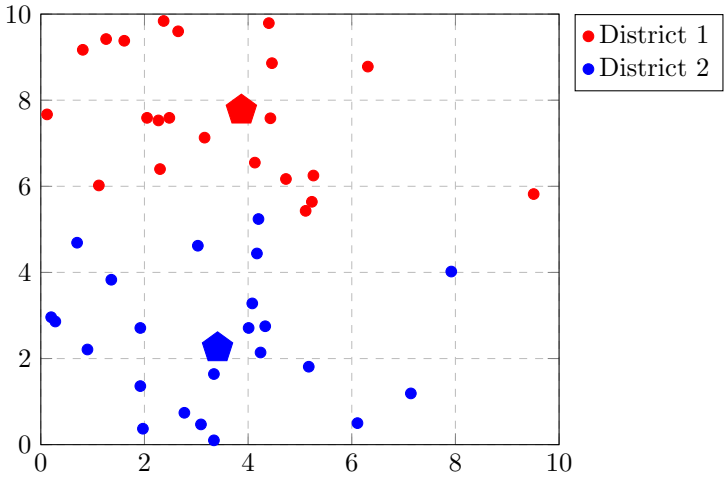


(b) ED

Figure A.3: The weekly schedules for week 3 from the best solution in an instance of 100_1.

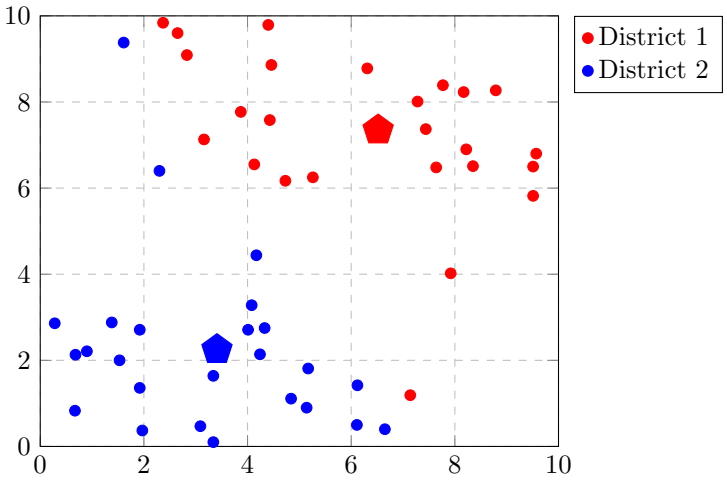


(a) NoDis

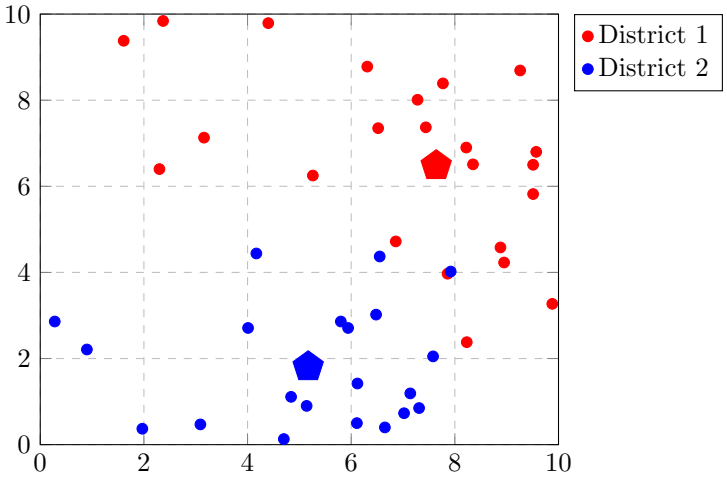


(b) ED

Figure A.4: The weekly schedules for week 4 from the best solution in an instance of 100_1.

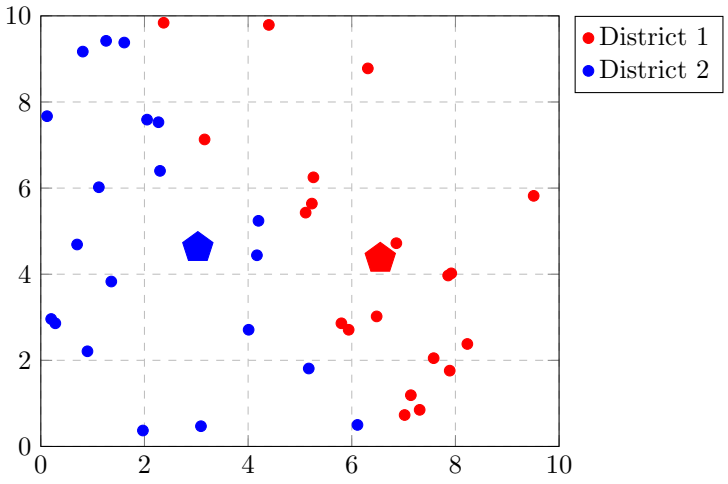


(a) NoDis

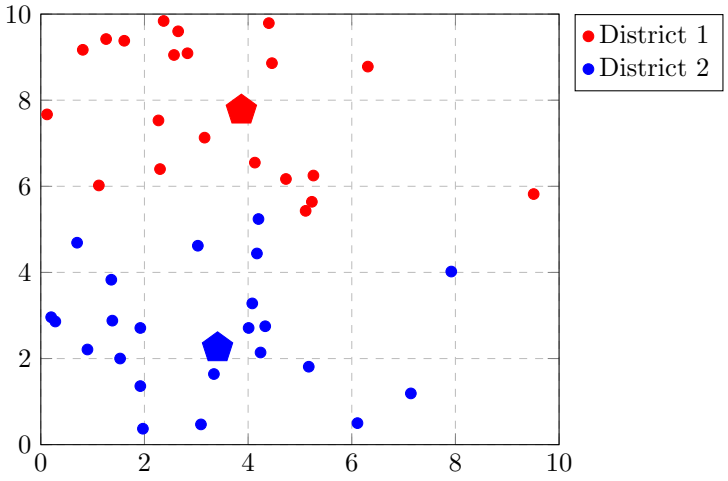


(b) ED

Figure A.5: The weekly schedules for week 5 from the best solution in an instance of 100_1.

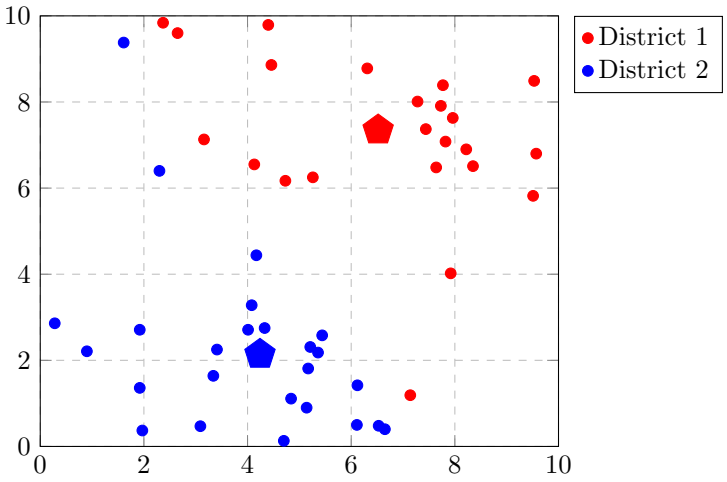


(a) NoDis

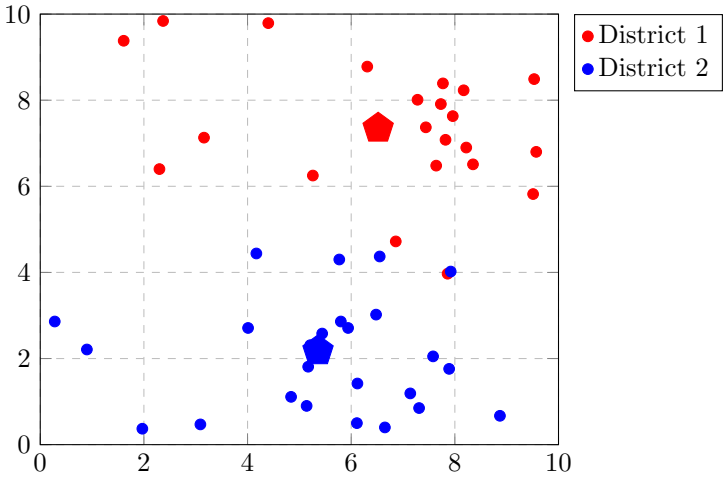


(b) ED

Figure A.6: The weekly schedules for week 6 from the best solution in an instance of 100_1.

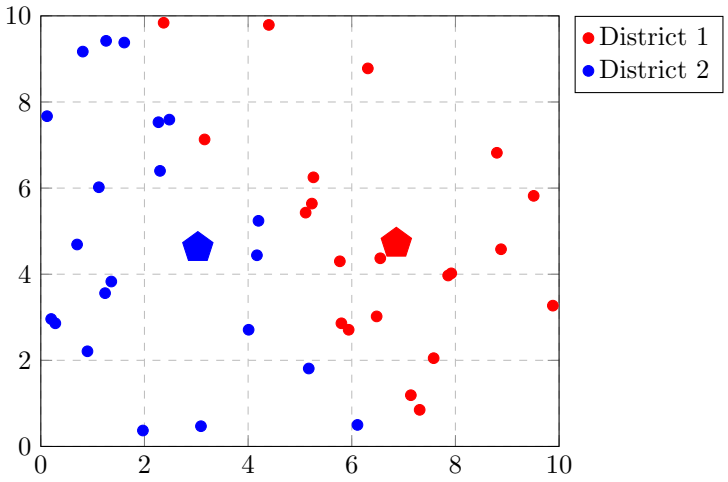


(a) NoDis

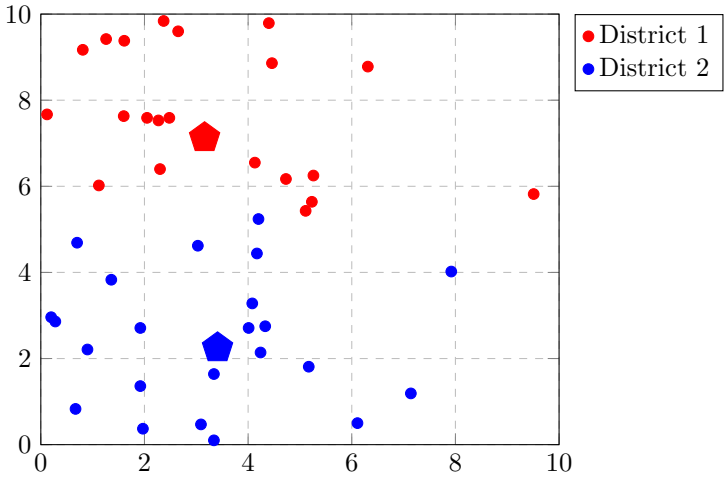


(b) ED

Figure A.7: The weekly schedules for week 7 from the best solution in an instance of 100_1.



(a) NoDis



(b) ED

Figure A.8: The weekly schedules for week 8 from the best solution in an instance of 100_1.

Bibliography

Adulyasak, Y., Cordeau, J.-F. & Jans, R. (2015), 'Benders decomposition for production routing under demand uncertainty', *Operations Research* **63**(4), 851–867.

Alrajhi, K., Thompson, J. & Padungwech, W. (2016), Tabu search hybridized with multiple neighborhood structures for the frequency assignment problem, *in* 'International Workshop on Hybrid Metaheuristics', Springer, pp. 157–170.

Angulo, G., Ahmed, S. & Dey, S. S. (2016), 'Improving the integer l-shaped method', *INFORMS Journal on Computing* **28**(3), 483–499.

Arthur, D. & Vassilvitskii, S. (2007), 'k-means++: The advantages of careful seeding, 1027–1035', *Society for Industrial and Applied Mathematics* .

Bação, F., Lobo, V. & Painho, M. (2005), 'Applying genetic algorithms to zone design', *Soft Computing* **9**(5), 341–348.

Bard, J. F. & Jarrah, A. I. (2009), 'Large-scale constrained clustering for rationalizing pickup and delivery operations', *Transportation Research Part B: Methodological* **43**(5), 542–561.

Bender, M., Kalcsics, J., Nickel, S. & Pouls, M. (2018), 'A branch-and-price algorithm for the scheduling of customer visits in the context of multi-period service territory design', *European Journal of Operational Research* **269**(1), 382–396.

Bender, M., Meyer, A., Kalcsics, J. & Nickel, S. (2016), 'The multi-period service territory design problem – an introduction, a model and a heuristic approach', *Transportation Research Part E: Logistics and Transportation Review* **96**, 135 – 157.

Benders, J. F. (1962), 'Partitioning procedures for solving mixed-variables programming problems', *Numerische Mathematik* **4**(1), 238–252.

Birge, J. & Louveaux, F. (1997), *Introduction to stochastic programming*, Springer, New York.

Blais, M., Lapierre, S. D. & Laporte, G. (2003), 'Solving a home-care districting problem in an urban setting', *Journal of the operational research society* **54**(11), 1141–1147.

Boland, N., Fischetti, M., Monaci, M. & Savelsbergh, M. (2016), 'Proximity Benders: a decomposition heuristic for stochastic programs', *Journal of Heuristics* **22**(2), 181–198.

Botton, Q., Fortz, B., Gouveia, L. & Poss, M. (2013), 'Benders decomposition for the hop-constrained survivable network design problem', *INFORMS Journal on Computing* **25**(1), 13–26.

Bozkaya, B., Erkut, E., Haight, D. & Laporte, G. (2011), 'Designing new electoral districts for the city of edmonton', *Interfaces* **41**(6), 534–547.

Bozkaya, B., Erkut, E. & Laporte, G. (2003), 'A tabu search heuristic and adaptive memory procedure for political districting', *European journal of operational research* **144**(1), 12–26.

Browdy, M. H. (1990), 'Simulated annealing: an improved computer model for political redistricting', *Yale Law & Policy Review* **8**(1), 163–179.

Butsch, A., Kalcsics, J. & Laporte, G. (2014), 'Districting for arc routing', *INFORMS Journal on Computing* **26**(4), 809–824.

Campbell, A. M. & Hardin, J. R. (2005), 'Vehicle minimization for periodic deliveries', *European Journal of Operational Research* **165**(3), 668–684.

Ceschia, S., Di Gaspero, L. & Schaerf, A. (2011), 'Tabu search techniques for the heterogeneous vehicle routing problem with time windows and carrier-dependent costs', *Journal of Scheduling* **14**(6), 601–615.

Codato, G. & Fischetti, M. (2006), 'Combinatorial Benders' cuts for mixed-integer linear programming', *Operations Research* **54**(4), 756–766.

Contreras, I., Cordeau, J.-F. & Laporte, G. (2011*a*), 'Benders decomposition for large-scale uncapacitated hub location', *Operations Research* **59**(6), 1477–1490.

Contreras, I., Cordeau, J.-F. & Laporte, G. (2011*b*), 'Stochastic uncapacitated hub location', *European Journal of Operational Research* **212**(3), 518 –528.

Contreras, I., Cordeau, J.-F. & Laporte, G. (2012), 'Exact solution of large-scale hub location problems with multiple capacity levels', *Transportation Science* **46**(4), 439–459.

Cooper, L. (1964), 'Heuristic methods for location-allocation problems', *SIAM Review* **6**(1), 37–53.

Cordeau, J.-F., Pasin, F. & Solomon, M. (2006), 'An integrated model for logistics network design', *Annals of Operations Research* **144**(1), 59–82.

Cortinhal, M. J. & Captivo, M. E. (2003), 'Upper and lower bounds for the single source capacitated location problem', *European journal of operational research* **151**(2), 333–351.

D'Amico, S. J., Wang, S.-J., Batta, R. & Rump, C. M. (2002), 'A simulated annealing approach to police district design', *Computers & Operations Research* **29**(6), 667–684.

Dayarian, I., Crainic, T. G., Gendreau, M. & Rei, W. (2016), 'An adaptive large-neighborhood search heuristic for a multi-period vehicle routing problem', *Transportation Research Part E: Logistics and Transportation Review* **95**, 95–123.

de Assis, L. S., Franca, P. M. & Usberti, F. L. (2014), 'A redistricting problem applied to meter reading in power distribution networks', *Computers & Operations Research* **41**, 65–75.

de Camargo, R., Miranda Jr, G., Ferreira, R. & Luna, H. (2009), 'Multiple allocation hub-and-spoke network design under hub congestion', *Computers and Operations Research* **36**(12), 3097–3106.

de Camargo, R. S., de Miranda, G. & Ferreira, R. P. (2011), 'A hybrid outer-approximation/Benders decomposition algorithm for the single allocation hub location problem under congestion', *Operations Research Letters* **39**(5), 329–337.

de Sá, E. M., de Camargo, R. S. & de Miranda, G. (2013), 'An improved Benders decomposition algorithm for the tree of hubs location problem', *European Journal of Operational Research* **226**(2), 185–202.

de Sá, E. M., Morabito, R. & de Camargo, R. S. (2018), 'Benders decomposition applied to a robust multiple allocation incomplete hub location problem', *Computers and Operations Research* **89**, 31–50.

Drexl, A. & Haase, K. (1999), 'Fast approximation methods for sales force deployment', *Management Science* **45**(10), 1307–1323.

Elizondo-Amaya, M. G., Ríos-Mercado, R. Z. & Díaz, J. A. (2014), 'A dual bounding scheme for a territory design problem', *Computers & operations research* **44**, 193–205.

Fernández, E., Kalcsics, J. & Nickel, S. (2013), 'The maximum dispersion problem', *Omega* **41**(4), 721–730.

Fernández, E., Kalcsics, J., Nickel, S. & Ríos-Mercado, R. Z. (2010), 'A novel maximum dispersion territory design model arising in the implementation of the WEEE-directive', *Journal of the Operational Research Society* **61**(3), 503–514.

Fischetti, M., Ljubić, I. & Sinnl, M. (2017), 'Redesigning Benders decomposition for large-scale facility location', *Management Science* **63**(7), 2146–2162.

Fischetti, M., Salvagnin, D. & Zanette, A. (2010), 'A note on the selection of Benders' cuts', *Mathematical Programming* **124**(1-2), 175–182.

Fleischmann, B. & Paraschis, J. N. (1988), 'Solving a large scale districting problem: a case report', *Computers & Operations Research* **15**(6), 521–533.

Fontaine, P. & Minner, S. (2018), 'Benders decomposition for the hazmat transport network design problem', *European Journal of Operational Research* **267**(3), 996–1002.

Forman, S. L. & Yue, Y. (2003), Congressional districting using a TSP-based genetic algorithm, *in* 'Genetic and Evolutionary Computation Conference', Springer, pp. 2072–2083.

Fortz, B. & Poss, M. (2009), 'An improved Benders decomposition applied to a multi-layer network design problem', *Operations Research Letters* **37**(5), 359–364.

Francis, P. M., Smilowitz, K. R. & Tzur, M. (2008), The period vehicle routing problem and its extensions, *in* 'The vehicle routing problem: latest advances and new challenges', Springer, pp. 73–102.

Gelareh, S., Neamatian Monemi, R. & Nickel, S. (2015), 'Multi-period hub location problems in transportation', *Transportation Research Part E* **75**, 67–94.

Gendreau, M., Hertz, A. & Laporte, G. (1994), 'A tabu search heuristic for the vehicle routing problem', *Management science* **40**(10), 1276–1290.

Gendreau, M. & Potvin, J.-Y. (2019), Tabu search, *in* M. Gendreau & J.-Y. Potvin, eds, 'Handbook of Metaheuristics', Springer International Publishing, Cham, pp. 37–55.

Gendron, B., Garroppo, R. G., Nencioni, G., Scutellà, M. G. & Tavanti, L. (2013), 'Benders decomposition for a location-design problem in green wireless local area networks', *Electronic Notes in Discrete Mathematics* **41**, 367 – 374.

Gendron, B., Potvin, J.-Y. & Soriano, P. (2003), 'A tabu search with slope scaling for the multicommodity capacitated location problem with balancing requirements', *Annals of Operations Research* **122**(1-4), 193–217.

Gendron, B., Scutellà, M. G., Garroppo, R. G., Nencioni, G. & Tavanti, L. (2016), 'A branch-and-Benders-cut method for nonlinear power design in green wireless local area networks', *European Journal of Operational Research* **255**(1), 151 – 162.

Geoffrion, A. M. (1970*a*), 'Elements of large-scale mathematical programming: Part I: Concepts', *Management Science* **16**(11), 652–675.

Geoffrion, A. M. (1970*b*), 'Elements of large scale mathematical programming: Part II: Synthesis of algorithms and bibliography', *Management Science* **16**(11), 676–691.

Geoffrion, A. M. (1972), 'Generalized Benders decomposition', *Journal of Optimization Theory and Applications* **10**(4), 237–260.

Geoffrion, A. M. & Graves, G. W. (1974), 'Multicommodity distribution system design by Benders decomposition', *Management Science* **20**(5), 822–844.

Gliesch, A., Ritt, M. & Moreira, M. C. (2018), A multistart alternating tabu search for commercial districting, *in* 'European Conference on Evolutionary Computation in Combinatorial Optimization', Springer, pp. 158–173.

Gross, J. L. & Yellen, J. (2003), *Handbook of graph theory*, CRC press.

Haugland, D., Ho, S. C. & Laporte, G. (2007), 'Designing delivery districts for the vehicle routing problem with stochastic demands', *European Journal of Operational Research* **180**(3), 997–1010.

Hess, S. W. & Samuels, S. A. (1971), 'Experiences with a sales districting model: criteria and implementation.', *Management Science* **18**(4), 41–54.

Hess, S. W., Weaver, J. B., Siegfeldt, H. J., Whelan, J. N. & Zitlau, P. A. (1965), 'Nonpartisan political redistricting by computer', *Operations Research* **13**(6), 998–1006.

IBM (2021), 'CPXERR_NOT_FOR_BENDERS: 2004 problem not compatible with Benders'. **URL:** *https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0/ilog.odms.cplex.help/refcallablelibrary/macros/CPXERR_NOT_FOR_BENDERS.html*

IBM ILOG CPLEX Optimization Studio CPLEX User's Manual (2017), 12.7 edn.

Irnich, S., Schneider, M. & Vigo, D. (2014), Chapter 9: Four variants of the vehicle routing problem, *in* 'Vehicle Routing: Problems, Methods, and Applications, Second Edition', SIAM, pp. 241–271.

Jin, J., Crainic, T. G. & Løkketangen, A. (2012), 'A parallel multi-neighborhood cooperative tabu search for capacitated vehicle routing problems', *European Journal of Operational Research* **222**(3), 441–451.

Kalcsics, J. & Ríos-Mercado, R. Z. (2019), Districting problems, *in* G. Laporte, S. Nickel & F. Saldanha da Gama, eds, 'Location Science', Springer International Publishing, pp. 705–743.

Kazan, O., Dawande, M., Sriskandarajah, C. & Stecke, K. E. (2012), 'Balancing perfectly periodic service schedules: An application from recycling and waste management', *Naval Research Logistics (NRL)* **59**(2), 160–171.

Kergosien, Y., Gendreau, M. & Billaut, J.-C. (2017), 'A Benders decomposition-based heuristic for a production and outbound distribution scheduling problem with strict delivery constraints', *European Journal of Operational Research* **262**(1), 287–298.

Kulturel-Konak, S., Norman, B. A., Coit, D. W. & Smith, A. E. (2004), 'Exploiting tabu search memory in constrained problems', *INFORMS Journal on Computing* **16**(3), 241–254.

Lei, H., Laporte, G. & Guo, B. (2011), 'The capacitated vehicle routing problem with stochastic demands and time windows', *Computers & Operations Research* **38**(12), 1775–1783.

Lei, H., Laporte, G. & Guo, B. (2012), 'Districting for routing with stochastic customers', *EURO Journal on Transportation and Logistics* **1**(1-2), 67–85.

Lei, H., Laporte, G., Liu, Y. & Zhang, T. (2015), 'Dynamic design of sales territories', *Computers & Operations Research* **56**, 84–92.

Lodish, L. M. (1975), 'Sales territory alignment to maximize profit', *JMR, Journal of Marketing Research (pre-1986)* **12**(000001), 30.

Magnanti, T. L. & Wong, R. T. (1981), 'Accelerating Benders decomposition: Algorithmic enhancement and model selection criteria', *Operations Research* **29**(3), 464–484.

Mariel, K. & Minner, S. (2017), 'Benders decomposition for a strategic network design problem under nafta local content requirements', *Omega* **68**, 62–75.

McDaniel, D. & Devine, M. (1977), 'A modified Benders' partitioning algorithm for mixed integer programming', *Management Science* **24**(3), 312–319.

Mehrotra, A., Johnson, E. & Nemhauser, G. (1998), 'An optimization bases heuristic for political districting', *Management Science* **44**(8), 1100–1114.

Mercier, A., Cordeau, J.-F. & Soumis, F. (2005), 'A computational study of Benders decomposition for the integrated aircraft routing and crew scheduling problem', *Computers and Operations Research* **32**(6), 1451–1476.

Mercier, A. & Soumis, F. (2007), 'An integrated aircraft routing, crew scheduling and flight retiming model', *Computers & Operations Research* **34**(8), 2251–2265.

Michel, L. & Van Hentenryck, P. (2004), 'A simple tabu search for warehouse location', *European Journal of Operational Research* **157**(3), 576–591.

Mourgaya, M. & Vanderbeck, F. (2007), 'Column generation based heuristic for tactical planning in multi-period vehicle routing', *European Journal of Operational Research* **183**(3), 1028–1041.

Nanry, W. P. & Barnes, J. W. (2000), 'Solving the pickup and delivery problem with time windows using reactive tabu search', *Transportation Research Part B: Methodological* **34**(2), 107–121.

Naoum-Sawaya, J. & Elhedhli, S. (2010), 'A nested Benders decomposition approach for telecommunication network planning', *Naval Research Logistics (NRL)* **57**(6), 519–539.

Naoum-Sawaya, J. & Elhedhli, S. (2013), 'An interior-point Benders based branch-and-cut algorithm for mixed integer programs', *Annals of Operations Research* **210**(1), 33–55.

Nemhauser, G. L. & Wolsey, L. A. (1988), *Integer and Combinatorial Optimization*, John Wiley & Sons, New York.

Núñez-del-Toro, C., Fernández, E., Kalcsics, J. & Nickel, S. (2016), 'Scheduling policies for multi-period services', *European Journal of Operational Research* **251**(3), 751 – 770.

Oliveira, F., Grossmann, I. E. & Hamacher, S. (2014), 'Accelerating Benders stochastic decomposition for the optimization under uncertainty of the petroleum product supply chain', *Computers & Operations Research* **49**, 47–58.

Papadakos, N. (2008), 'Practical enhancements to the Magnanti–Wong method', *Operations Research Letters* **36**(4), 444–449.

Papadakos, N. (2009), 'Integrated airline scheduling', *Computers and Operations Research* **36**(1), 176–195.

Paquette, J., Cordeau, J.-F., Laporte, G. & Pascoal, M. M. (2013), 'Combining multicriteria analysis and tabu search for dial-a-ride problems', *Transportation Research Part B: Methodological* **52**, 1–16.

Pearce, R. H. & Forbes, M. (2018), 'Disaggregated Benders decomposition and branch-and-cut for solving the budget-constrained dynamic uncapacitated facility location and network design problem', *European Journal of Operational Research* **270**(1), 78–88.

Pishvaee, M., Razmi, J. & Torabi, S. (2014), 'An accelerated Benders decomposition algorithm for sustainable supply chain network design under uncertainty: A case study of medical needle and syringe supply chain', *Transportation Research Part E* **67**, 14–38.

Rahmaniani, R., Crainic, T. G., Gendreau, M. & Rei, W. (2018), 'Accelerating the Benders decomposition method: Application to stochastic network design problems', *SIAM Journal on Optimization* **28**(1), 875–903.

Rahmaniani, R., Crainic, T., Gendreau, M. & W., R. (2017), 'The Benders decomposition algorithm: A literature review', *European Journal of Operational Research* **259**(3), 801 – 817.

Ribeiro, G. M. & Laporte, G. (2012), 'An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem', *Computers & operations research* **39**(3), 728–735.

Ríos-Mercado, R. Z. (2016), 'Assessing a metaheuristic for large-scale commercial districting', *Cybernetics and Systems* **47**(4), 321–338.

Rios-Mercado, R. Z. & Escalante, H. J. (2016), 'GRASP with path relinking for commercial districting', *Expert Systems with Applications* **44**, 102–113.

Ríos-Mercado, R. Z. & Fernández, E. (2009), 'A reactive GRASP for a commercial territory design problem with multiple balancing requirements', *Computers & Operations Research* **36**(3), 755–776.

Ríos-Mercado, R. Z., Maldonado-Flores, J. & González-Velarde, J. (2017), Tabu search with strategic oscillation for improving recollection assignment plans of waste electric and electronic equipment, *in* 'Technical Report', PISIS-2017-01, Universidad Autónoma de Nuevo León San Nicolás de los Garza.

Ríos-Mercado, R. Z. & Salazar-Acosta, J. C. (2011), A GRASP with strategic oscillation for a commercial territory design problem with a routing budget constraint, *in* 'Mexican International Conference on Artificial Intelligence', Springer, pp. 307–318.

Rochat, Y. & Taillard, É. D. (1995), 'Probabilistic diversification and intensification in local search for vehicle routing', *Journal of heuristics* **1**(1), 147–167.

Ropke, S. & Pisinger, D. (2006*a*), 'An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows', *Transportation Science* **40**(4), 455–472.

Ropke, S. & Pisinger, D. (2006*b*), 'A unified heuristic for a large class of vehicle routing problems with backhauls', *European Journal of Operational Research* **171**(3), 750–775.

Salazar-Aguilar, M. A., Ríos-Mercado, R. Z. & González-Velarde, J. L. (2013), 'GRASP strategies for a bi-objective commercial territory design problem', *Journal of Heuristics* **19**(2), 179–200.

Salazar-Aguilar, M., Ríos-mercado, Roger, Z. & Cabrera-ríos, M. (2011), 'New models for commercial territory design', *Networks and Spatial Economics* **11**(3), 487–507.

Santibanez-Gonzalez, E. D. & Diabat, A. (2013), 'Solving a reverse supply chain design problem by improved Benders decomposition schemes', *Computers & Industrial Engineering* **66**(4), 889–898.

Santoso, T., Ahmed, S., Goetschalckx, M. & Shapiro, A. (2005), 'A stochastic programming approach for supply chain network design under uncertainty', *European Journal of Operational Research* **167**(1), 96–115.

Sherali, H. D. & Lunday, B. J. (2013), 'On generating maximal nondominated Benders cuts', *Annals of Operations Research* **210**(1), 57–72.

Sherali, H. D. & Soyster, A. L. (1983), 'Preemptive and nonpreemptive multi-objective programming: Relationship and counterexamples', *Journal of Optimization Theory and Applications* **39**(2), 173–186.

Shirabe, T. (2009), 'Districting modeling with exact contiguity constraints', *Environment and Planning B: Planning and Design* **36**(6), 1053–1066.

Soto, M., Sevaux, M., Rossi, A. & Reinholz, A. (2017), 'Multiple neighborhood search, tabu search and ejection chains for the multi-depot open vehicle routing problem', *Computers & Industrial Engineering* **107**, 211–222.

Steiner, M. T. A., Datta, D., Neto, P. J. S., Scarpin, C. T. & Figueira, J. R. (2015), 'Multi-objective optimization in partitioning the healthcare system of Parana state in Brazil', *Omega* **52**, 53–64.

Stenger, A., Vigo, D., Enz, S. & Schwind, M. (2013), 'An adaptive variable neighborhood search algorithm for a vehicle routing problem arising in small package shipping', *Transportation Science* **47**(1), 64–80.

Taşkın, Z. C. & Cevik, M. (2013), 'Combinatorial Benders cuts for decomposing IMRT fluence maps using rectangular apertures', *Computers and Operations Research* **40**(9), 2178–2186.

Tang, L., Jiang, W. & Saharidis, G. K. D. (2013), 'An improved Benders decomposition algorithm for the logistics facility location problem with capacity expansions', *Annals of Operations Research* **210**(1), 165–190.

Tapia-Ubeda, F. J., Miranda, P. A. & Macchi, M. (2018), 'A generalized Benders decomposition based algorithm for an inventory location problem with stochastic inventory capacity constraints', *European Journal of Operational Research* **267**(3), 806–817.

Tavares-Pereira, F., Figueira, J. R., Mousseau, V. & Roy, B. (2007), 'Multiple criteria districting problems', *Annals of Operations Research* **154**(1), 69–92.

Üster, H. & Agrahari, H. (2011), 'A Benders decomposition approach for a distribution network design problem with consolidation and capacity considerations', *Operations Research Letters* **39**(2), 138–143.

Vatsa, A. K. & Jayaswal, S. (2016), 'A new formulation and Benders decomposition for the multi-period maximal covering facility location problem with server uncertainty', *European Journal of Operational Research* **251**(2), 404–418.

Wei, B. C. & Chai, W. Y. (2004), 'A multiobjective hybrid metaheuristic approach for GIS-based spatial zoning model', *Journal of Mathematical Modelling and Algorithms* **3**(3), 245–261.

Wei, W. & Liu, C. (1983), 'On a periodic maintenance problem', *Operations Research Letters* **2**(2), 90–93.

Wu, X., Yan, S., Wan, X. & Lü, Z. (2016), 'Multi-neighborhood based iterated tabu search for routing and wavelength assignment problem', *Journal of Combinatorial Optimization* **32**(2), 445–468.

Xia, Y., Fu, Z., Pan, L. & Duan, F. (2018), 'Tabu search algorithm for the distance-constrained vehicle routing problem with split deliveries by order', *PloS one* **13**(5), e0195457.

Yaghini, M., Karimi, M. & Rahbar, M. (2013), 'A hybrid metaheuristic approach for the capacitated p-median problem', *Applied soft computing* **13**(9), 3922–3930.

Zoltners, A. A. & Sinha, P. (2005), 'The 2004 ISMS practice prize winner - sales territory design: Thirty years of modeling and implementation', *Marketing Science* **24**(3), 313–331.

Žulj, I., Kramer, S. & Schneider, M. (2018), 'A hybrid of adaptive large neighborhood search and tabu search for the order-batching problem', *European Journal of Operational Research* **264**(2), 653–664.